LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SÓLO PROGRAMADORES

Revista especializada para usuarios de PC

AÑO 3. Nº 20
1250 PTAS.

ARGENTINA 9'50 $
CHILE 3000 $

El análisis semántico

UNIX Aplicaciones distribuidas

Creación de un compilador

VISUAL C++ 4.0
La librería de clases MFC

CD-ROM DE REGALO
LAS DEMOS DEL CONCURSO Y UTILIDADES PARA INTERNET

WEB

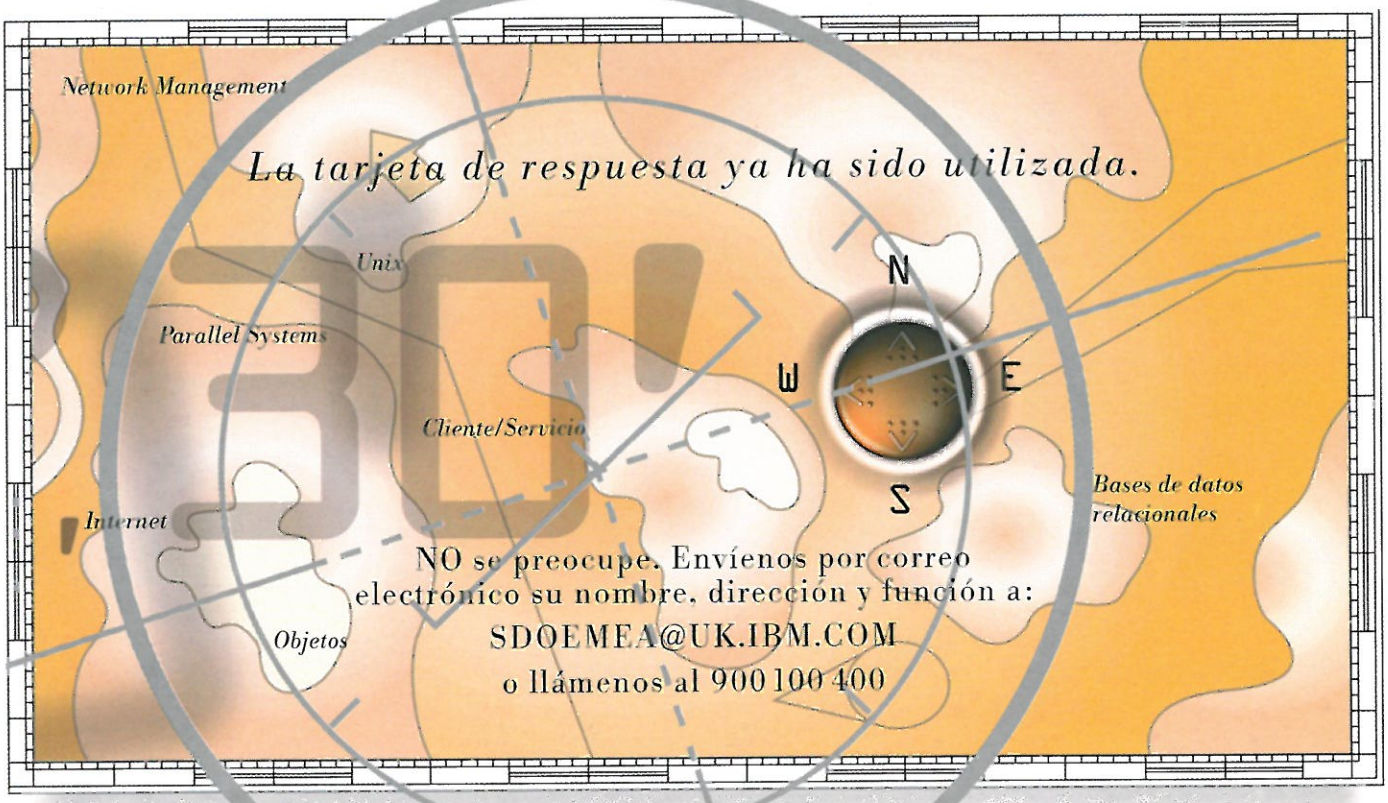
CGI: EL INTERFAZ DEL FUTURO

Y además:

- CURSO DE DEMOS: Comenzamos con las 3D
- PROGRAMACIÓN: Estructura Árbol
- FORMATOS GRÁFICOS: PNG
- PROGRAMACIÓN DE LA VGA
- LA MEMORIA EXPANDIDA
- VISUAL BASIC 4.0

8 413042 303299 00020

TOWER COMMUNICATIONS, S.R.L.



Cómo lograr y mantener una posición líder en el vertiginoso negocio del desarrollo de software.

Para desarrollar programas se necesitan buenas ideas así como los conocimientos y las herramientas necesarios para tranformarlas en soluciones excepcionales. La clave para llegar y mantenerse en una posición destacada es el acceso fácil a la información necesaria. Por eso IBM ha creado D_Mail: un nuevo servicio de información para empresas y profesionales desarrolladores de software.

D_Mail le facilita la más reciente información sobre el impacto potencial de las últimas innovaciones, análisis detallados sobre tecnologías clave, el panorama de oportunidades profesionales para el mundo de las soluciones de software y mucho más. ¡Sin coste para usted!

Sólo tiene que devolvernos cumplimentada la tarjeta adjunta, y nosotros le ayudaremos a trazar el rumbo correcto hacia el futuro.

IBM

Soluciones para nuestro pequeño mundo

SÓLO PROGRAMADORES
Revista práctica para usuarios de PC

ABRIL 1996. Número 20
SÓLO PROGRAMADORES es una publicación de Tower Communications

Editor
Antonio M. Ferrer Abelló

Director Editorial
Mario de Luis García

Director Financiero
Francisco García Díaz de Liaño

Director de Producción
Carlos Peropadre

Directora Comercial
Carmina Ferrer

Director
Mario de Luis García

Redactor Jefe
Carlos Doral Pérez

Coordinador Técnico
Eduardo Toribio Pazos

Editor Técnico
Luis Martin Caballero

Publicidad
Mª Eugenia González

Colaboradores
Fernando de la Villa, Fernando J. Echevarrieta, Pedro Antón. Juan M. Martín, Luis Martín, José María Peco, José Carlos Remiro, Agustín Guillén, Francisco Monteagudo, Ignacio Cea, Daniel Navarro, Jorge del Río, Enrique De Alarcón, Pere Ridau, Jordi Freixenet, Enrique Torres

Maquetación
Fernando García Santamaría

Tratamiento de Imagen
María Arce Giménez

Servicio Técnico
Óscar Rodríguez

Servicios Informáticos
Digital Dreams Multimedia, S.L.

Secretaría de Redacción
Rosa Arroyo

Suscripciones
Erika de la Riva

Redacción, Publicidad y Administración
C/ Aragoneses, 7
28100 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86

Filmación
Filma Dos S.L.

Impresión
G.D.B.

Distribución
SGEL

La revista SÓLO PROGRAMADORES no tiene por qué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados.
El editor prohibe expresamente la reproducción total o parcial de los contenidos de la revista sin su autorización escrita.

Depósito legal: M-26827-1994
ISSN: 1134-4792

EDITORIAL

PC´S O GRANDES SISTEMAS

Tradicionalmente, los grandes sistemas han estado discriminados por los programadores, que preferían dedicarse al mundo de los microordenadores. Sin embargo, las grandes empresas siempre defendieron la conveniencia de la coexistencia entre los dos mundos, incluso cuando las apariencias indicaban lo contrario. El propio gigante azul, IBM, pasó por momentos de apuro por la insistencia en mantener vivo su AS/400.

Tras unos años de oscurantismo y discriminación, parece que los grandes sistemas vuelven a renacer. Para comprobar tal afirmación, basta con echar un vistazo a las ofertas de trabajo en las páginas de la prensa diaria. Por fin, parece que comienza a encontrarse el sitio apropiado para cada sistema, sin necesidad de que ambos tengan que entrar en colisión. A pesar de ello, siempre habrá gente tanto a favor como en contra de alguno de los sistemas.

El problema principal con el que se enfrentan los grandes sistemas es la dificultad que encuentran para la expansión dentro del mundo de los informáticos. Hoy en día resulta realmente sencillo el acceso a algún cursillo sobre cualquiera de las herramientas utilizadas por los microordenadores, como por ejemplo, Word, Excel, Word Perfect, Windows, MS-DOS, etc. Sin embargo, resulta realmente extraño ver academias anunciando cursos de operador de AS/400 o MVS, o de programación en DB2 o COBOL/400.
Esto supone un problema añadido a la expansión de los grandes sistemas ya que, en la mayoría de ocasiones, el acceso a un curso sobre cualquier aspecto relacionado con estos supone para el interesado un fuerte desembolso, que en la mayoría de las ocasiones no se ve recompensado en el mercado laboral. Esta recompensa no es tanto por falta de puestos de trabajos, sino más bien por la remuneración de los mismos, así como las exigencias de las empresas para con los posibles candidatos. El problema radica en que, por norma general, los cursos sobre estos sistemas los imparten las propias empresas. Esto supone una fuerte inversión para ellas, resultando lógico que exijan a los pretendientes un compromiso de no abandonar la empresa, una vez adquiridos los conocimientos.

Resulta comprensible la postura tanto de la empresas, negándose dar una formación que pueda ser aprovechada por la competencia, como la de la comunidad de informáticos, prefiriendo el trabajo en entornos de microordenadores, más baratos y accesibles para el gran público.

Con el afán de tocar todos los temas relacionados con el ámbito de la programación, en nuestra revista se viene incluyendo un curso sobre estos grandes sistemas. Con él se trata de descubrir un mundo que, tradicionalmente, ha estado apartado por los programadores, y que esperamos pueda ser todo un descubrimiento para alguno de nuestros lectores.

Continuando con esa política empresarial, el presente número toca múltiples aspectos de la programación. Empezando con el trabajo a bajo nivel, con el curso sobre creación de Demos, el manejo de la VGA o la utilización de la memoria EMS. También tienen cabida los nuevos, y cada vez más extendidos, lenguajes visuales, con las dos herramientas más extendidas como son Visual C++ y Visual Basic. Otra de las partes que esta resultando ampliamente seguida por los lectores es lo relacionado con las páginas Web, que tras finalizar con el lenguaje HTML se introduce en la más moderna tecnología con la interfaz CGI, la última herramienta aparecida para la creación de aplicaciones para Web.

Por último y para completar este número, el CD ROM con las mejores DEMOS realizadas por todos nuestros lectores.

SUMARIO

NÚMERO 20

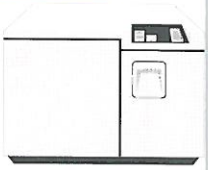
10 **WORLD WIDE WEB**
Con este artículo la serie da un giro para adentrarse en el mundo de la programación de aplicaciones para Web, comenzando en primer lugar, por una introducción a la indiscutible interfaz CGI.

16 **CURSO DE PROGRAMACIÓN**
En esta nueva entrega se presentan ideas básicas sobre el tratamiento de árboles binarios, además se incluye una descripción sobre los árboles avl adecuados para realizar procesos de ordenación y búsqueda.

22 **CURSO DE UNIX**
Seguimos con el diseño de aplicaciones distribuidas. Una vez expuestos los conceptos necesarios para la programación con RPCs de SUN, se completa este mes el tema con el desarrollo de una aplicación distribuida.

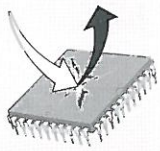
28 **CÓMO HACER UNA DEMO**
Comenzamos con las 3D. En este número de Abril se explicarán las fórmulas de rotación y proyección en tres dimensiones. En los próximos meses aprenderemos todo lo necesario para nuestras demos.

33 **GRANDES SISTEMAS**
Continuando con el tema iniciado el pasado mes, este le toca el turno a los métodos de almacenamiento y recuperación de la información en ficheros con organización directa y particionada.

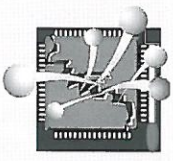


38 **SISTEMA OPERATIVO LINUX**
Se presenta este mes el último de los artículos dedicados a la implementación de drivers sobre Linux. En esta ocasión, se expone el funcionamiento de los drivers para dispositivos de bloques.

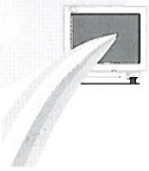
45 **CREACIÓN DE UN COMPILADOR**
El compilador está ya prácticamente perfilado, esto quiere decir que aunque queden muchas decisiones por tomar, sólo las referentes al análisis semántico afectan al lenguaje.



52 **CURSO DE VISUAL BASIC 4.0**
Los formularios son el soporte básico de las aplicaciones realizadas con Visual Basic. Cada una de las ventanas que componen la aplicación será diseñada a partir de un formulario.

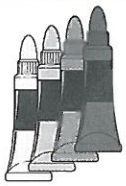


57 **PROGRAMACIÓN DE LA VGA**
En este artículo, enmarcado en la programación avanzada del adaptador gráfico VGA, nos ocuparemos de la gestión de los modos gráficos nativos del mismo, es decir, los modos basados en planos de bits.



63 **MEMORIA EXPANDIDA**
Gracias a la memoria LIM-EMS, las aplicaciones DOS pueden superar la limitación de los 640 Kbytes de memoria disponible bajo el mismo, esto es necesario para poder crear aplicaciones bajo DOS en modo real, mínimamente profesionales.

67 **FORMATOS GRÁFICOS**
Ya hace tiempo que Compuserve, propietaria del formato GIF, anunció su intención de cobrar derechos a aquellas firmas que incluyeran imágenes en este formato en sus programas. La comunidad informática respondió: crear un formato gráfico alternativo, el PNG.

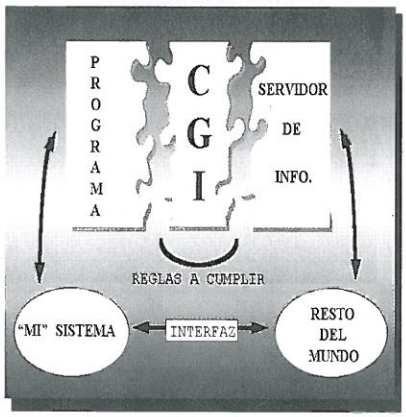


73 **VISUAL C++ 4.0**
Desde la aparición de la librería de clases MFC para programación en Windows, ha ido progresivamente evolucionando hacia un entorno de desarrollo sencillo y potente, donde la reutilización de código y la programación visual apartan progresiva y rápidamente la tortuosa programación en C del API.

79 **CONTENIDO DEL CD-ROM**
Este mes en nuestro CD incluimos la mayor parte de las demos participantes en nuestro primer concurso de Demos e Infografía, además una colección de utilidades shareware y los fuentes correspondientes de los artículos de este número.

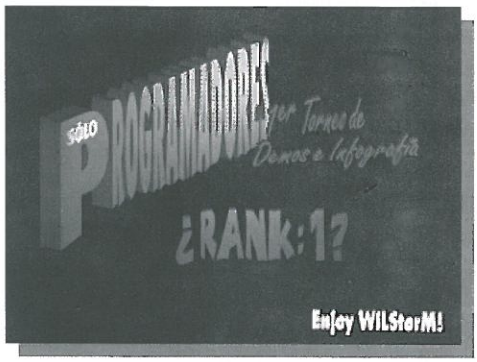
81 **CORREO DEL LECTOR**
Es el espacio dedicado a la resolución de los problemas surgidos a nuestros lectores en los diversos aspectos de la programación.

EN PORTADA

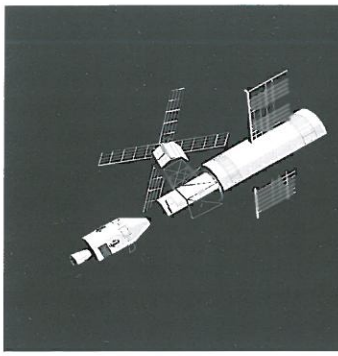


La interfaz CGI es el título de la nueva entrega de la serie dedicada al mundo del Web. Es nuestro artículo de portada en Abril, además destacado Visual C++, se pretende que los lectores que sigan esta sección sean capaces dentro de unos meses de crear sus propias aplicaciones profesionales mediante esta poderosa herramienta.

DESTACADO



El concurso de Demos tuvo su veredicto el mes pasado, y en nuestro CD de regalo incluimos la mayor parte de ellas. Por otra lado cabe destacar las secciones de Linux, este mes con el estudio de los drivers para dispositivos de bloques, Cómo hacer una Demo en la cual a partir de esta entrega se entra de lleno en el mundo de las 3D y el avance de todas las habituales secciones de nuestra revista como son Unix, Curso de Programación, Visual Basic 4.0, etc.



SÓLO PROGRAMADORES

NOTICIAS

Intel presenta el primer controlador USB

La corporación Intel realizó una demostración del funcionamiento de los periféricos de PC y telefonía *USB* (*Universal Serial Bus*) en la Conferencia de Desarrolladores *USB* celebrada en Santa Clara. Los productos presentados se basan en el controlador Intel 82930A, el primero de una serie de controladores compatibles con *USB* para periféricos de PC. Este controlador permite a los productos de telefonía y periféricos de PC comunicar con cualquier *host USB*, como aquellos basados en los nuevos chips PCI 430VX/HX de Intel. El controlador es compatible con la versión 1.0 de la especificación *USB* desarrollada conjuntamente por Compaq, Digital, IBM PC Co., Intel, Microsoft, NEC y Northern Telecom.

El controlador 82930A de Intel es un dispositivo de altas prestaciones de 8 bits y 12 MHz con memoria y amplia capacidad de direccionamiento, baja potencia, poco ruido y soporte eficaz de idiomas de alto nivel. Además soporta transferencias de datos isócronos y asíncronos e incluye un motor de interfaz de bus serie con descodificación y generación de paquetes *USB*.

La producción en volumen está prevista para el final del segundo trimestre del presente año y el precio en cantidades de 10.000 unidades para *OEMs* es de 6 dólares por unidad.

Para más información:
Intel Corporation Iberia
Tel: (91) 308 25 52
Fax: (91) 310 54 60
World Wide Web: http://www.intel.com

Microsoft presenta Internet Information Server

Microsoft ha anunciado la disponibilidad de *Internet Information Server* (*IIS*), el producto con el que Microsoft se incorpora al mercado de servidores para *World Wide Web*. Se trata del primer servidor de Web integrado con el sistema operativo de red Windows NT Server y ofrece un rendimiento superior a otras soluciones existentes en el mercado.

Internet Information Server permite a las empresas la creación de direcciones Web en Internet e Infovía y dentro de las redes corporativas o intranets. Está integrado con la familia de productos *Microsoft BackOffice* por lo que permite el desarrollo de toda una nueva generación de aplicaciones para Web y habilita el acceso en formato Web de las aplicaciones existentes de bases de datos y mensajes.

Las pruebas realizadas por el *NSTL* (*National Software Testing Laboratories*) muestran que *Internet Information Server* supera a otros servidores de Web para Windows NT en un 250 por ciento como mínimo, conforme a las pruebas estándar. Los tests realizados con Webstone 1.1 también señalan que es más rápido que otros servidores Web para UNIX de mayor precio.

El producto puede descargarse de Internet sin coste alguno en la página de información de Microsoft (*http ://www.microsoft.com/infoserv*).

Para más información:
Elena Arroyo
Spanish Executive Information Service
Tel: (91) 308 43 15
Fax: (91) 310 38 76

Nueva versión de Apple Media Tool

Apple Computer Inc. Ha anunciado recientemente la versión 2.0 de *Apple Media Tool* y de su producto complementario *Apple Media Tool Programming Environment*. Se trata de una nueva herramienta de autor multiplataforma con facilidad de uso basado en objetos que permite a los usuarios crear títulos interactivos sin necesidad de *scripts*. Una vez creados los proyectos en la plataforma Macintosh se pueden crear versiones ejecutables para Macintosh, Power Macintosh y Windows con un simple clic.

Apple Media Tool Programming Environment está diseñado para los desarrolladores que están buscando introducirse en la programación multimedia orientada a objetos. La versión 2.0 incluye el subconjunto de *MPW* (*Macintosh Programmerís Workbench*) distribuido sin cargo adicional. *Apple Media Tool Programming Environment* trabaja en conjunción con el *MPW Shell* para depurar los proyectos multimedia.

La nueva versión de *Apple Media Tool* incorpora características como soporte de películas QuickTime VR, paletas de color personalizables y AppleScript.

Para más información:
SMPS
François Connan
Tel: (93) 237 12 08

Nueva familia de conjuntos de chips de 4ª generación

Intel Corporation ha anunciado una nueva familia de conjuntos de chips PCI formada por los conjuntos 430HX y 430VX. El conjunto 430HX está orientado a los PCs profesionales basados en el procesador Pentium y el 430VX está dirigido a los PCs domésticos y de pequeños negocios basados en el mismo procesador. Estos productos proporcionan una mayor escalabilidad y mejoras a través del nuevo *Concurrent PCI* y del soporte de la iniciativa de la industria *USB* (*Universal Serial Bus*), además de mejoras de las prestaciones globales del sistema. El conjunto PCI 430HX de Intel está formado por el Controlador del Sistema 82439HX (TXC) en encapsulado BGA de 324 pines y el Acelerador PCI ISA IDE 82371SB (PIIX3) en encapsulado PQFP de 208 pines. Ambos conjuntos incluyen un *Bridge host PCI* que proporciona funciones como el interfaz de bus host de 64 bits, un interfaz PCI de 32 bits y un interfaz para la memoria principal de 64 bits. Para cantidades de 1.000 unidades, el conjunto 430HX con encapsulado BGA cuesta 37,50 dólares y el conjunto 430VX con encapsulado QFP cuesta 33 dólares.

Para más información:
Intel Corporation Iberia
Tel: (91) 308 25 52
Fax: (91) 310 54 60
World Wide Web: http://www.intel.com

ExpOracle'96

Durante los días 21 y 22 del pasado mes de febrero tuvo lugar la tercera edición de la exposición informática ExpOracle en el Palacio de Congresos y Exposiciones de Madrid. ExpOracle ha estado dedicada a la tecnología de Oracle para la gestión de la información y ha contado con la presencia de diversas empresas del sector informático como Unisys, Novell, Siemens Nixdorf, Hewlett Packard, Bull, IBM, Silicon Graphics, Sun Microsystems y Digital.

En los diferentes stands de Oracle Ibérica se pudieron ver las novedades con respecto a *Oracle Warehouse*, una solución que permite el manejo de datos heterogéneos, la integración de cualquier fuente y el empleo de cualquier herramienta con el fin de facilitar la toma de decisiones empresariales. También se mostró *Oracle Workgroup/2.000*, la gama de productos económicos de Oracle para entornos PC y de grupos de trabajo compuesto por software de conectividad, bases de datos, middleware y herramientas para programadores.

La exposición también incluyó una muestra de los productos OLAP de Oracle, dirigidos a gerentes, directores generales, analistas y responsables de departamentos que permite analizar grandes volúmenes de datos y sintetizar la información necesaria para adoptar decisiones rápidas según los datos.

En ExpOracle'96 se celebraron de forma totalmente gratuita 32 ponencias sobre diversos temas como Internet, multimedia, la evolución cliente/servidor, reingeniería y migración de aplicaciones.

Satelec'96

Entre los días 4 y 7 de marzo de 1996 se celebraron las XXI Jornadas de Comunicaciones, Electrónica e Informática en la Escuela Superior de Ingenieros de Telecomunicación de Madrid con el apoyo oficial del Rectorado de la UPM, Fundesco, el Colegio Oficial de Ingenieros de Telecomunicación (COIT) y la Dirección General de Telecomunicaciones (DGTel), entre otros.

En Satelec'96 las actividades se estructuraron en dos grandes grupos. Por una parte la exposición técnica formada por los stands de las entidades participantes, y por otra parte un ciclo de conferencias sobre diversos temas de actualidad.

Las Jornadas han contado con la colaboración, entre otros, de Andersen Consulting, Airtel, Tower Communications, Telefónica de España, Argentaria y Alcatel.

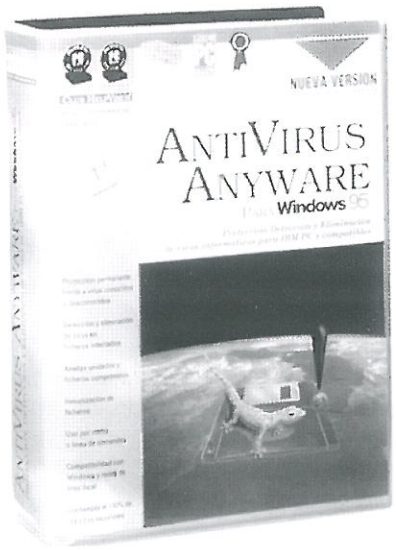
En el desarrollo del ciclo de conferencias participaron empresas de diversa índole y se trataron temas como la liberalización de las comunicaciones, la tarjeta universitaria, las comunicaciones, la teleenseñanza, Internet y el teletrabajo.

Primer Congreso Nacional de Usuarios de Internet

En el Palacio de Congresos y Exposiciones de Madrid tuvo lugar, entre el 14 y el 16 de febrero, el Primer Congreso Nacional de Usuarios de Internet por iniciativa de la Asociación de Usuarios de Internet. Durante la celebración del este congreso se desarrolló de forma paralela una exposición de acceso libre en la que participaron algunas de las empresas relacionadas con el sector.

Internet World'96 contó con la presencia de una treintena de empresas con varios stands que permitían el acceso a Internet. A lo largo de los tres días se celebró un ciclo de conferencias dedicadas a una gran variedad de materias íntimamente relacionadas con la red de redes. Se trataron diversos temas como la tecnología, el mercado e impacto estratégico, la seguridad, y servicios y educación.

El Primer Congreso Nacional de Usuarios de Internet proporcionó una amplia y clara imagen del sector, sin aportar novedades en los aspectos más técnicos, y ha sentado las bases para un acontecimiento que previsiblemente se repetirá de nuevo.



NOVEDADES

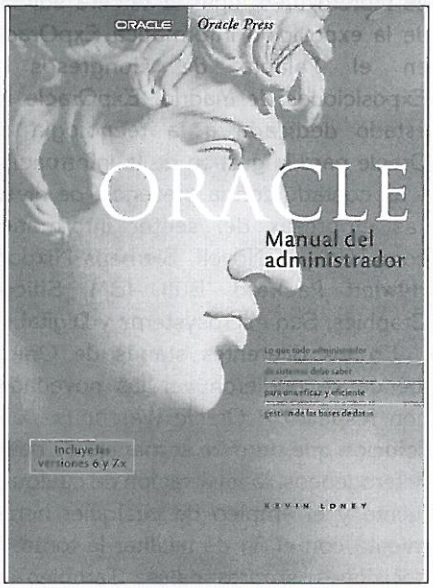
Antivirus Anyware para Windows 95

La empresa española Anyware ha lanzado la última versión del Antivirus Anyware para el sistema operativo Windows 95. El programa protege al ordenador de la actividad de todos los virus conocidos y no conocidos mediante un sistema exclusivo de verificación de integridad de los ficheros y de protección de zonas críticas del equipo. El Antivirus Anyware queda residente en memoria desde el momento de su instalación y está totalmente en castellano.

La versión Windows 95 del antivirus proporciona funciones de análisis de disquetes, discos duros, CD-ROM y unidades en red, protección permanente y automática desde el arranque de la máquina, y eliminación de los virus localizados. Además permite la realización de un análisis manual en cualquier momento.

Al igual que el resto de los productos de la empresa, el paquete Antivirus Anyware para Windows 95 cuenta con un servicio "Hotline" de ayuda al usuario en las cuestiones relacionadas con el producto, así como un servicio permanente de recepción de consultas 24 horas al día todos los días del año a través de fax, modem y correo electrónico. Además, Anyware cuenta con un servicio "Club Helpvirus" que proporciona al usuario una serie de ventajas como envío de actualización de disquetes, información de nuevos productos y nuevas versiones.

LIBROS



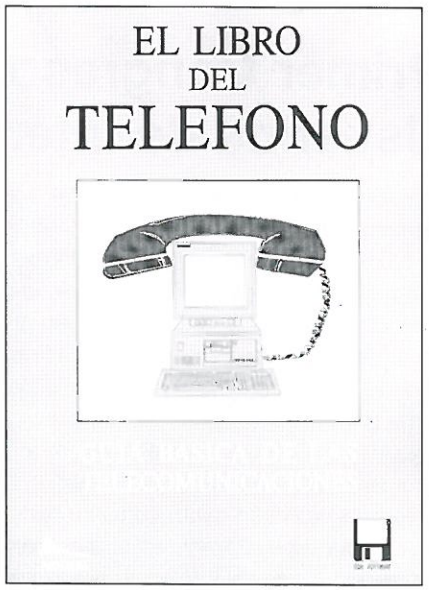
Oracle Manual del administrador

Se trata de una obra dirigida exclusivamente a los administradores de bases de datos Oracle en la que se presenta toda la información necesaria para gestionar de forma eficiente estos sistemas. El contenido del libro se encuentra organizado en cinco partes bien diferenciadas como la arquitectura de las bases de datos, la gestión de las bases de datos, los paquetes de soporte y Oracle en red. En ellas se tocan temas tan dispares como la seguridad, la gestión de bases de datos distribuidas y la supervisión de varias bases de datos. La última parte del libro está formada por más de 140 páginas de apéndices.

Editorial: McGraw-Hill
Autor: Kevin Loney
630 páginas
Idioma: Castellano
Precio: 4.900 pesetas (IVA incluido)

El libro del teléfono

Bajo este curioso título se esconde una completa guía que desvela todos los secretos de un tema tan apasionante como es la telecomunicación. La obra comienza con una introducción a los sistemas de telecomunicación, la telefonía, la red telefónica y la transmisión de datos para entrar de lleno en las redes y los servicios de telecomunicación. Algunos de los temas tratados son las redes punto a punto, la conmutación de circuitos, la Red Digital de Servicios Integrados, la videocomunicación y el teletrabajo. Junto con el libro se puede adquirir el programa Tesacom de Telefónica, sin coste adicional alguno.



Editorial: PROGENSA
Autores: J. Espallargas, F. Limonche y P. Robles
476 páginas
Idioma: Castellano
Precio: 5.750 pesetas

MENSAJES

Me fascina eso de que en la red local puedas mandar pequeños mensajes de una máquina a otra. En la sala de Desarrollo, decenas de nuevos mensajes se crean cada día, vuelan por los cables de la red apurando su más que efímera existencia, y se matasellan en los monitores como mosquitos aplastados contra el parabrisas. Unos son automáticos y previsibles, fáciles de ignorar como un buen mayordomo: la impresora HPLASER del nodo CHARLIE ha finalizado el trabajo de impresión. Otros son fedatarios únicos del trabajo de los administradores de red, esta gente constante y desconocida como las termitas: Desenganchaos del disco W:, vamos a reconfigurar el servidor de Administración. Otros nos llegan del compañero de al lado, y refrescan ocasionalmente el tedio de las pantallas con un chiste breve o un comentario cruel ante la melodía que alguien tararea con escasa fortuna sobre el repiquetear de las teclas. Éste es un cuento pequeño sobre mensajes.

Rafa es uno de estos tipos que no acaba de aterrizar, un chaval de veintidós años que ingresó en la compañía con contrato de seis meses. Ocupa, junto con cuatro chicas y otro compañero, un pequeño espacio limitado por mamparas, un remanso diminuto en un lugar de paso de la planta de Desarrollo, por el que siempre está pasando gente con prisa. Programa bien, tiene madera de buen técnico y, piensan los más experimentados, podría llegar lejos si adquiere algo más de mano izquierda, aprende a salir a comer con el grupo correcto y pierde esa conmovedora ignorancia del quién es quién. Rafa es un chico tímido, un corazón solitario que no ha conocido un solo amor correspondido. Esconde la cabeza detrás de su monitor para rehuir las miradas de sus cuatro compañeras, y se sonroja cuando Nacho, su único compañero, dispara un chiste malicioso. Pero a Rafa le gustan los mensajes de red, quizá porque no conocen el rubor ni el tartamudeo, y disfruta intercambiando criptogramas o pequeños retos de lógica, clamando pena de muerte contra el responsable de la canción de los setenta que suena en el hilo musical, o preguntando, a quien se lo pueda decir, cuál es la función que devuelve el puntero a la cola de mensajes del proceso actual.

Un martes ajetreadísimo, de una semana de ésas a las que no se ve principio ni final, un mensaje pequeño como una polilla se estampó en su pantalla:

Message from ROSA, Tue 96-04-11 12:00

>Te quiero, Rafa.

Press Escape to continue

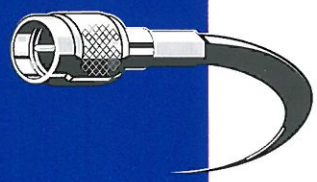
No tenía la menor importancia que ROSA fuera el nombre de su propio ordenador, puesto que el campo del remitente se puede alterar con un sencillo truco, y no era la primera vez que le honraban mensajes de Juan Pablo II o de Hillary Clinton. Tampoco era ninguna novedad que las chicas o su compañero le dedicaran una breve y socarrona declaración de amor a cambio del préstamo de un boli o la localización de un listado perdido. Pero en esa sucesión de caracteres había algo de una sinceridad absolutamente violenta. En esa cadena ASCII había algo imposible de malinterpretar, algo lleno de una certeza inquietante y perentoria. En la absurda medida en que se pueda decir que un mensaje de red tenga caligrafía, ese mensaje estaba escrito en serio, allí había unas letras pequeñitas y apretadas que tenían que salir de dentro de un corazón, no de unas ganas de cachondeo. Rafa optó por ponerse rojo como un tomate, mirar en derredor y ver cuatro caras femeninas absortas en sus pantallas. Lo peor de todo es que Nacho, su único compañero, estaba desplazado en un cliente, lo que dejaba cerrada toda vía de huida. Durante el resto del día no hubo más mensajes, pero Rafa produjo diez veces menos líneas de código que de costumbre, y todas contuvieron errores.

El miércoles, de nuevo a las doce en punto, recibió un brevísimo extracto de un poema de Pedro Salinas que hablaba del tiempo, del amor y de lo inevitable. Durante las dos semanas siguientes, siempre a la hora del ángelus, fueron llegando nuevos mensajes, largos o cortos, apasionados, tiernos, coquetos y hasta incendiarios. Sus cuatro compañeras seguían herméticamente simpáticas y cordiales, pero Rafa no podía deshacerse de la sensación de que las cuatro le miraban de una manera distinta, de que cada una de esas cuatro miradas contenía algo de ocultación y algo de promesa. ¿Pero quién? Rafa pasó esos quince días con sus noches en un estado febril. El último de esos días se levantó de golpe, con el rostro resuelto, como si un sueño le hubiera dado la respuesta. Se desayunó con dos cápsulas de vitaminas, reunió la totalidad de sus fuerzas y al dar las nueve invitó a un café a Blanca, la bajita y morena de los ojos grandes, y se encontró con que ella le hablaba como se habla a un hombre a quien se puede querer, no como a un programador patoso con contrato basura y manchas de rotulador en las manos. Y resultó que no hablaron de C ni de tarjetas de sonido, sino de las pasiones silenciosas de cada uno, de los miedos personales, de la vida que se les escapaba entre los dedos. Ese día los mensajes cesaron, y Blanca y Rafa empezaron una relación que aún sigue creciendo. Lo cierto es que hacen una pareja maravillosa.

Ése fue el día en que Nacho, el bufón que tenía por compañero, borró por fin del ordenador el fichero *batch* que había colado en los ordenadores de Rafa y Blanca, y que se había ocupado de lanzarles las regulares flechas de Cupido aun en su ausencia. Lo hizo con una sonrisa malévola de fauno y con la satisfacción del trabajo bien hecho.

LA INTERFAZ CGI: PRIMER CONTACTO

Fernando J. Echevarrieta



Con este artículo la serie da un giro para adentrarse en el mundo de la programación de aplicaciones para web, comenzando en primer lugar, por una introducción a la indiscutible interfaz CGI.

Al fin ha llegado el momento de comenzar con el apasionante mundo de la programación CGI. En este artículo se realizarán algunas consideraciones sobre la interfaz CGI, los lenguajes empleados en la programación de *scripts* CGI y los requisitos necesarios para desarrollarlos. Se realizarán unos primeros y sencillos *scripts* y se dejarán las puertas abiertas para profundizar en los próximos capítulos. A lo largo de éstos, se hará también alguna pausa en el camino para indicar las nuevas extensiones que se vayan incorporando al HTML, como la definición de *client-side imagemaps* o las nuevas etiquetas de Microsoft.

PRESENTACIÓN DE LA INTERFAZ

CGI (*Common Gateway Interface*) es una interfaz entre aplicaciones externas y servicios de información. Hasta el momento, se ha utilizado (con acogida abrumadora) únicamente como pasarela hacia servidores WWW que hablan el protocolo HTTP. Sin embargo, una de las causas de su lenta estandarización es que se ha planteado como pasarela independiente del servicio y de la aplicación.

Un documento HTML, es algo estático, permanente, lo que no se adecúa a las necesidades de interactividad, acceso a información en constante actualización, consulta a bases de datos o seguimiento, control o recuperación de resultados de un determinado proceso. Por ello, a veces es necesario acceder a una información que se está generando "en tiempo real". Así pues, es necesario contar con algún tipo de pasarela como la que define CGI.

La interfaz define una forma cómoda y simple de ejecutar programas que se encuentran en la máquina en la que se aloja el servidor. Para el cliente presenta una ventaja en el aspecto de la seguridad, ya que no tendrá que ejecutar ningún programa de efectos desconocidos en su sistema local. Desde el punto de vista humano, además de eliminar la necesidad de aprendizaje, se resuelven los problemas de mantenimiento, operación y distribución de clientes ya que el acceso se realizará a través de cualquier cliente estándar de WWW y la comunicación se realizará según el protocolo HTTP.

Sin embargo, aunque en el servidor ya no será necesario permitir el acceso a extraños, un *webmaster* deberá saber que tener un programa CGI equivale a que "TODO EL MUNDO podrá ejecutar un programa en MI sistema", por lo que sí será necesario tener en cuenta algunos aspectos elementales sobre seguridad. El más básico suele ser la necesidad de incluir los programas CGI en un directorio determinado reservado al efecto, al que generalmente se le llama *cgi-bin*, directorio que no suele ser accesible a todos los usuarios. Cuando un servidor de WWW recibe como petición un documento alojado en este directorio, entenderá que se trata de un programa ejecutable.

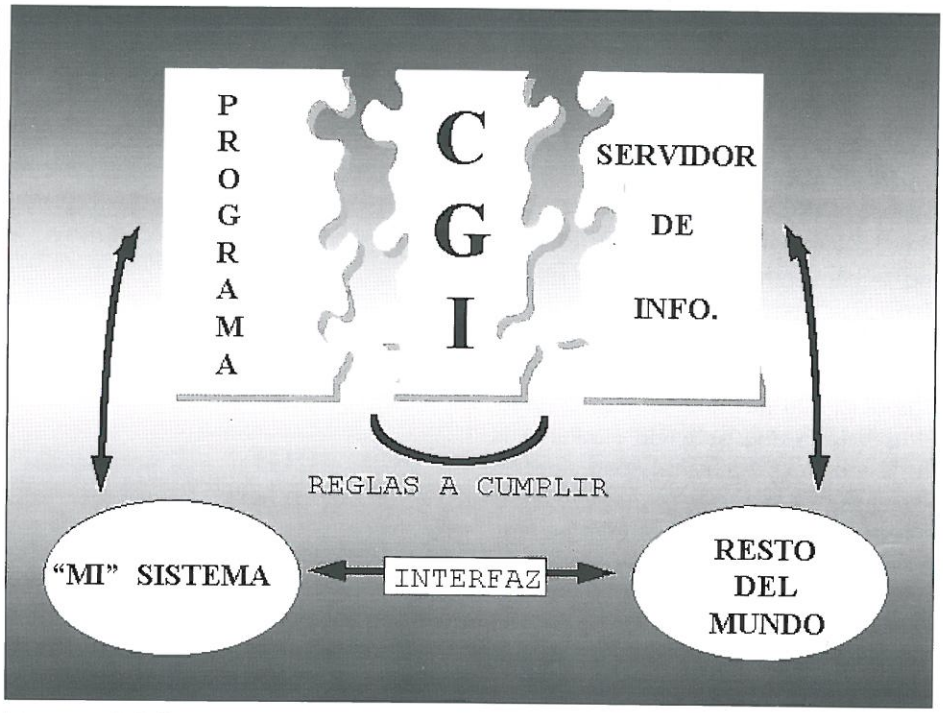
Pero CGI tampoco es la panacea universal. Algunas voces señalan como inconvenientes que plantea la necesidad de ejecutar programas en el servidor, de los cuales es necesario tener un copia por cliente ya que no permite hacer uso de recursos compartidos y, al tratarse de una pasarela, no es posible disponer de un control directo de la comunicación. Por ello se plantean

nuevas alternativas, como el *NSAPI* de Netscape Corporation y, por supuesto, *Java*. Pero éste es otra historia que no se puede pasar por alto, debido a que ha sido ya adoptado por Netscape y Microsoft, lo que asegura que se convertirá en uno de los estándares más fuertes.

CGI EN WINDOWS

Windows no dispone de un intérprete de comandos nativo. Por ello, un *back-end* para CGI debe ser un programa ejecutable adicional al servidor. Para comunicarse con este programa se podría haber utilizado OLE o DDE, pero se ha optado por una comunicación a través de ficheros intermedios. Éste es el único hecho diferencial de la especificación CGI para servidores Windows. La información sobre esta interfaz y el servidor Windows que se facilitó con el número del mes pasado se puede encontrar en *http://www.city.net/win-httpd*.

Por lo demás, la especificación de la interfaz es la misma. Debido a ello, a que la mayoría de los servidores WWW se ejecutan sobre UNIX y a que sobre este sistema se puede realizar una explicación de forma general, los ejemplos de estos artículos se realizarán sobre UNIX, aunque como se verá son perfectamente válidos (el código específico, por supuesto no lo será, pero el uso de la interfaz, sí) para cualquier otro sistema y, también, para aquellos usuarios que instalen y configuren el *back-end* de Windows. Por otra parte, con el Linux que se facilita periódicamente con la revista, es posible disponer de un servidor con el que desarrollar y depurar sin necesidad de red.



Esquema de la interfaz CGI

LO NECESARIO PARA DESARROLLAR CGIs

Si el lector no es *webmaster* de su sistema, deberá ponerse en contacto con el mismo para solicitarle el acceso al directorio de ejecutables o para que configure un directorio perteneciente al lector como directorio alternativo de CGIs. En cualquier caso, la mejor opción es convertirse en *webmaster*. En el número anterior (19) de Sólo Programadores se facilitó un servidor para Linux y un programa de instalación automático, así como las instrucciones para realizar una instalación manual del servidor del *W3 Consortium* en cualquier otro sistema UNIX. Este servidor se puede obtener de forma gratuita para múltiples plataformas en *http://www.w3.org*.

Aún sin contar con un servidor de información será posible, aunque no tan emocionante, desarrollar y probar programas CGI. Por una parte, habrá que simular "a mano", los datos que el servidor facilita al CGI. Por otra, habrá que obtener la respuesta del CGI y verificar que es correcta. Si bien, puede parecer carente de sentido desarrollar CGIs que no van a ser instalados en un servidor, la finalidad didáctica es indiscutible. A través de los ejemplos se irán indicando los pasos.

CGI es una interfaz entre aplicaciones externas y servicios de información

LENGUAJES DE PROGRAMACIÓN CGI

CGI, como la propia palabra indica es una interfaz entre servidores de información y programas de aplicación. Por tanto, define una serie de reglas que deben cumplir tanto las aplicaciones como los servidores para hacer posible la presentación de resultados de programas ejecutables en tiempo real a través de servicios de información estandarizados. Por ello, se habla de *gateway* o pasarela entre una y otra dimensión. Al tratarse de una interfaz, no existe ningún tipo de dependencia con el lenguaje de programación empleado y, del mismo modo, no presentará mayor dificultad o facilidad realizar un programa CGI en un lenguaje u otro, salvo la intrínseca al propio lenguaje.

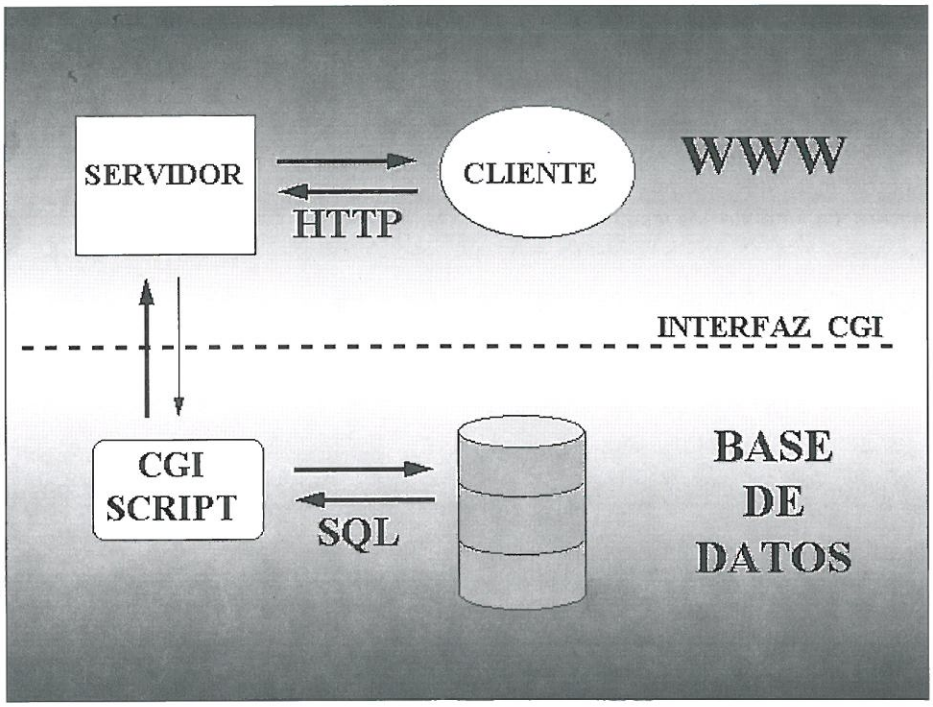
En principio, cualquier lenguaje es susceptible de ser utilizado para desarrollar programas CGI, (o *scripts* CGI, en el argot) ya sea interpretado o compilado. En "la red" existen aplicaciones CGI desarrolladas en C, C++, Fortran, Perl, Tcl, Visual Basic, AppleScript y cualquier shell de UNIX. A la hora de elegir un lenguaje de programación, mucha gente se decanta por lenguajes interpretados como *sh*, *tcl* y, sobre todo, *perl*, alegando su mayor facilidad de mantenimiento y depuración. Otra

"ventaja" de los lenguajes interpretados es que, por lo general, suelen ser de más alto nivel, con lo que no permiten realizar ciertas "maniobras" con la memoria y es más difícil dejar procesos sueltos descontrolados en el sistema. Por esto, muchos *webmasters* se niegan a admitir programas de usuario que no se encuentren realizados en ciertos lenguajes interpretados.

Por ello, uno de los lenguajes más utilizados y solicitados por las empresas es el *Perl* que, siendo interpretado, proporciona una potencia similar a la de C con una sintaxis muy parecida. En cualquier caso, si se conoce bien el interfaz y no se conoce el *Perl*, no será difícil realizar una defensa justificada ante una posible oferta de trabajo en la que se exija este lenguaje.

La contrapartida es que un lenguaje compilado es siempre mucho más rápido que uno interpretado. En el caso de CGI la velocidad de ejecución es importante, ya que habrá que sumar el tiempo de ejecución a la latencia de red y a la velocidad de transmisión, tiempos de espera que tendrá que sufrir el usuario del cliente.

En la mayoría de los casos, el lenguaje apropiado será el que dicte la compañía para la que se trabaja o, en su defecto, el que se conozca. En esta serie, se empleará en algunos ejemplos la shell de UNIX (interpretado) y, en otros, el lenguaje C (compilado). Si no programa en sh le será de gran ayuda consultar [Echeva-1].



Uno de los usos más extendidos de la interfaz es el acceso a bases de datos a través de WWW.

Disponer de un script CGI equivale a que "todo el mundo podrá ejecutar un programa en MI sistema"

EL PARADIGMA DE PROGRAMACIÓN CGI

La documentación acerca de programación CGI es más bien escasa y, hasta el momento, se encuentra únicamente en inglés. En Enero (8 de Enero de ¡1996!), D. Robinson de la Universidad de Cambridge, ha realizado el primer *draft* de RFC describiendo la interfaz, aunque se puede encontrar documentación desde mucho antes en el servidor del *W3 Consortium, http://www.w3.org/pub/WWW/CGI*. La interfaz CGI es fruto de las conversaciones que se han venido realizando a través de una lista de correo: la *www-talk* y en su mayor parte, se debe a Rob McCool, John Franks, Ari Luotonen, George Phillips y Tony Sanders. Pero lo que no figura en ningún medio escrito (al menos que conozca el autor) es que la programación CGI para WWW lleva asociada un paradigma propio al tratarse el HTTP de un protocolo de transmisión sin estado [Echeva-2]. Será necesario tener esto en cuenta a la hora de desarrollar aplicaciones, especialmente si son interactivas, por lo que habrá que disponer de algún agente externo (memoria de masa, demonios [Echeva-3]), encargado de almacenar este estado. El autor tratará de transmitir este paradigma a través de las aplicaciones que se realicen.

INVOCANDO UN CGI

La forma más simple de invocar un CGI es mediante una URL que apunte al fichero ejecutable, que se encontrará en el directorio reservado al efecto. El servidor, al recibir una petición dentro de este directorio comprenderá que se trata de un programa y lo ejecutará. Esto ocurrirá cada vez que se acceda al URL correspondiente. Por ejemplo, supóngase que el servidor se encuentra en *mimaquina.midominio.es*, el directorio de CGIs es *cgi-bin* y el programa CGI recibe el nombre de *no_se_que_hago*. Bastará abrir desde un browser de WWW el URL: *http://mimaquina.midominio.es/cgi-bin/no_se_que_hago*

Por supuesto, éste, como cualquier otro URL, puede ser un enlace dentro de un documento. De esta forma, es posible definir un panel de control remoto a través de WWW que contenga una serie de botones cada uno de los cuales active un programa diferente al ser pulsado. Sin embargo, esto no seria de mucha utilidad si el programa no pudiera devolver respuestas y datos.

"¡EL CGI ESTÁ TRATANDO DE DECIRNOS ALGO!"

Al ejecutarse un CGI, su salida estándar queda inmediatamente conectada con el servidor de información, que la hará llegar al cliente. Un programa CGI puede devolver cual-

quier tipo de información, desde un simple texto ASCII, a una imagen GIF, o JPEG o ... pasando por una animación, código HTML y un largo etcétera.

Un browser será incapaz de mostrar esta información de forma correcta si no conoce de qué naturaleza es, del mismo modo que se producirán efectos inesperados si se trata de abrir un fichero GIF con un editor de texto. Por ello, la interfaz CGI especifica una serie de cabeceras que los *scripts* pueden utilizar para proporcionar este tipo de información. Así, aunque la salida del *script* es un flujo de datos, se puede considerar dividido en cabeza y cuerpo, como ocurría con los documentos HTML. Entre las cabeceras, la más importante es la que especifica el contenido:

Content-type:

tras ella, debe indicarse el tipo MIME a que pertenece la información que será generada por el documento. El tipo se indica de la forma habitual *tipo/subtipo*. Por ejemplo:

Content-type: image/jpeg

Es extremadamente IMPORTANTE, dejar una línea en blanco entre las cabeceras CGI y el contenido de la información. Si no se hace así, el sistema no funcionará y el cliente no podrá acceder a la información.

Tras ello, el cuerpo de datos, podrá ser, como se indicaba, cualquier tipo de información. Por poner un ejemplo, supongamos que se desea realizar una aplicación que simplemente muestre una página HTML generada en tiempo real. Su salida deberá ser de la forma indicada en el *listado 1*.

LISTADO 1

```
Content-type: text/html

<HTML>
<HEAD>
<TITLE>Mi primer CGI</TITLE>
</HEAD>
<BODY>
Hola <B>mundo</B>!
</BODY>
</HTML>
```

LISTADO 2

```
#!/bin/sh
echo "Content-type: text/html"
echo ""
echo <HTML>
echo <HEAD>
echo <TITLE>Mi primer CGI</TITLE>
echo </HEAD>
echo <BODY>
echo Hola <B>mundo</B>!
echo </BODY>
echo </HTML>
```

LISTADO 3

```
main()
{
    printf("Content-type: text/html\n\n");
    printf("<HTML>\n");
    printf("<HEAD>\n");
    printf("<TITLE>Mi primer CGI</TITLE>\n");
    printf("</HEAD>\n");
    printf("<BODY>\n");
    printf("Hola <B>mundo</B>!\n");
    printf("</BODY>\n");
    printf("</HTML>\n");
}
```

Un posible programa en sh de UNIX para realizar esto sería el que se puede observar en el *listado 2*.

Y un posible programa en C que daría exactamente la misma salida sería el que aparece en el *listado 3*.

Con ello, se ve como es indiferente la utilización de uno u otro lenguaje para programar CGI. Obsérvese una vez más cómo se deja una línea en blanco entre la cabecera y la información. Si desde el browser, tras ejecutar el CGI, se observa la fuente del documento, aparecerá únicamente la parte correspondiente al código HTML, con lo que no habrá ninguna diferencia con lo que se hubiera observado si éste documento fuera un fichero HTML cualquiera. Imagine el lector las posibilidades que se le abren si en lugar de escribir "Hola mundo", el CGI escribiera la hora, la temperatura, la carga del sistema, los usuarios conectados... Piense que en ese documento HTML podría haber enlaces a otros documentos u otros programas CGI. Existirán tantas salidas diferentes posibles para un mismo programa como compleja sea su lógica.

PREGUNTE VD. ALLÍ

No sólo es posible devolver cierto tipo de información a través de un CGI, sino también un puntero a la información deseada. Por supuesto, este puntero será un URL. Para ello, se ha definido otra cabecera de CGI denominada *Location* que se emplea de la forma

Location: <URL>

Por ejemplo, la salida de un CGI podría ser:

Content-type: text/html
Location: http://highland.dit.upm.es:-8000

[Cuerpo]

Obsérvese una vez más la línea en blanco entre cabeceras y cuerpo. Lo habitual es que un CGI que responda con una cabecera *Location* no incluya cuerpo ni precise de cabecera *Content-type*. Sin embargo, aún es posible encontrar CGIs con una salida como ésta en previsión de la utilización de algún browser muy antiguo que no realice el salto automáticamente. Así, se podría incluir un cuerpo como:

<HTML>
<HEAD>
<TITLE>Enlace a otro documento</TITLE>
<HEAD>
<BODY>
Esa información esta disponible <A HREF=http://highland.dit.upm.es:8000>aquí</A>
</BODY>
</HTML>

Como repaso, se recuerda también al lector que es posible realizar un salto temporizado mediante la etiqueta *<META>* de HTML 3.0 que incorporan ya las versiones 2.x de Netscape y Mosaic [Echeva-4].

Es posible desarrollar y depurar CGIs aún sin servidor ni red

De esta forma, es posible realizar un CGI que tome decisiones sobre qué información mostrar, desde la más simpática, como mostrar una foto de día durante las horas diurnas y otra de noche durante las nocturnas, hasta las más profundas. El uso más común de esta cabecera se produce cuando se realizan movimientos de información de un servidor a otro.

Es fundamental dejar una línea en blanco entre las cabeceras CGI y el cuerpo de la información

CÓMO FUNCIONA UN CONTADOR WWW

Una de las posibilidades que más llaman la atención de los navegantes de Internet es la inclusión de contadores en páginas WWW. Pese a su espectacularidad, su funcionamiento es muy simple. Un contador de accesos a una página no es otra cosa que un CGI que se activa al ser accedida la página. Al ejecutarse toma un valor de un soporte de memoria "estable", lo incrementa y lo vuelve a almacenar en memoria "estable". Como salida, devuelve el valor de este contador.

Ante esto, surgen dos cuestiones: la primera, cómo hacer que se active ese contador al acceder a una página; la segunda, cómo mezclar la salida del CGI con el resto de la página. Para ello, existen dos respuestas triviales. Un método (*técnica A*) es hacer que toda la página sea un CGI, como en los ejemplos de los listados 2 y 3, o simplemente, un CGI que muestre el contador y luego devuelva por su salida estándar, una página HTML de disco.

Al ejecutarse un CGI su salida queda inmediatamente conectada con el servidor

La segunda, la más empleada y la única posible si no se puede crear CGIs (*técnica B*), es utilizar un CGI estándar cuya salida sea una imagen. Éste es el ejemplo más utilizado en Internet. El secreto está en que al tratarse de una imagen, puede ser incluida dentro de un documento mediante la etiqueta *<IMG>*. El browser, al procesar el documento, accederá inmediatamente al URL que le indique el atributo SRC de la etiqueta. Si este URL es el de un CGI, el programa se activará y su salida será conectada al browser. Al devolver el CGI un gráfico, éste se incorporará al documento como si de un gráfico cualquiera se tratara y el browser jamás sabrá que "le han engañado". Si el lector desea utilizar un contador de estas características puede utilizar el que se encuentra en *http://www.digits.com*.

Intencionadamente se ha denominado como "triviales" a estos dos métodos porque sí existe una forma de mezclar la salida de CGIs con documentos HTML a través de una técnica denominada *server parsing* (*técnica C*) que incorporan algunos servidores. Más adelante en la serie se explicará cómo realizar una emulación de esta técnica mediante CGIs de manera que sea posible incluir el resultado de uno o varios programas cualesquiera dentro de un mismo documento (sean o no CGIs) mediante una simple función desarrollada por el autor.

El ejemplo del *listado 4* simplemente muestra el número que indica el contador, pero podría mostrar una página completa como se mencionaba anteriormente (técnica A). Si el lector desea el contador simplemente para *accounting* (estadísticas de acceso) propio, sin necesidad de mostrar el valor, puede ser una buena solución que cualquiera de la imágenes de su página apunte al CGI. El CGI incrementará el contador y después mostrará la imagen (técnica mixta), lo que en shell puede ser tan sencillo como indicar:

dd if=<fichero_imagen>

LISTADO 4

```
#!/bin/sh

LOCK=/tmp/.counter_lock
FILE=/tmp/.counter_file

while [ -f "$LOCK" ]
do
  sleep 1
done

# Genera cerrojo
touch $LOCK

# Incrementa contador
COUNTER=`cat $FILE`
if [ "$COUNTER" = "" ]; then
  COUNTER=0
fi
COUNTER=`expr $COUNTER + 1`
echo $COUNTER > $FILE

# Libera cerrojo
rm $LOCK

# Salida del CGI
echo Content-type: text/html
echo ""
echo $COUNTER
```

Si el programa genera en tiempo real una imagen con dígitos, se tendrá lo que se ha comentado como técnica B.

JUNTOS PERO NO REVUELTOS

Es importante mencionar que a la hora de programar CGI es necesario considerar que se trabaja en un entorno concurrente en el que hay que tener especial cuidado en el acceso a datos compartidos, como puede ser el caso del contador en el que se podría dar el típico caso de la *actualización perdida*. Hay que considerar que mientras se está accediendo a ciertos datos, otro programa puede estar cambiándolos. Por ello, habrá que utilizar alguna técnica de programación concurrente, lo que escapa al contenido de esa serie. Como ejemplo más sencillo y utilizado se puede mencionar el empleo de *ficheros cerrojo* o *locks* que figura en el ejemplo del listado 4. Cuando un programa va a entrar en *zona crítica*,

comprueba si hay un cerrojo y en caso de que lo haya, espera. Cuando el cerrojo no esté presente lo creará, entrará en zona crítica y al salir lo eliminará. Este sistema garantiza la *exclusión mutua*, pero puede dar problemas de *inanición*, *deadlock*, o *bloqueo* si un programa muere antes de liberar el cerrojo que ha activado.

NO VA A FUNCIONAR

Hay una pregunta que antes o después se plantea todo desarollador de CGIs: "¿Por qué no funciona?". En las *news* se repite una y otra vez esta pregunta y, frecuentemente, las respuestas son siempre las mismas:

1. El CGI no devuelve datos por su salida estándar o no dispone de cabeceras. Es posible que en algunos casos, la salida de un CGI que no contiene cabeceras sea interpretada como *text/plain* lo que no es razón para no incluir un *Content-type*.

2. No hay una línea en blanco después de las cabeceras. Aún a riesgo de caer en la redundancia exagerada, se suele perder tanto tiempo por este detalle que conviene recordarlo una vez más.

3. El fichero correspondiente al CGI no es ejecutable o, en general, no tiene los permisos adecuados. Hay que recordar que, si bien el ejecutable que resulta de una compilación suele tener permisos de ejecución, al escribir la fuente para un programa interpretado, esto no suele ser así. En cualquier caso, conviene siempre ajustar los permisos manualmente por razones de seguridad [Echeva-5], [Echeva-6].

4. El fichero corresponde a un programa interpretado y no se ha indicado el intérprete. Este es otro de los errores que más tiempo hacen perder a los principiantes.

Dos cuestiones clave son cómo activar un CGI automáticamente y cómo incluir su salida en una página

Debido al funcionamiento de los ficheros de texto ejecutables [Echeva-1], el sistema debe conocer cuál será el intérprete que los ejecutará. Así, en el *listado 2*, la línea

#!/bin/sh

es necesaria. Si se elimina, el sistema devolverá un error al servidor al tratar de ejecutar el *script* aunque éste tenga los permisos ajustados de forma correcta. Esto se debe a que no sabrá qué intérprete lanzar para procesarlo.

CONCLUSIÓN

La verdadera potencia de los programas CGI será puesta de manifiesto cuando se les facilite información desde el exterior. El próximo mes se comenzará mostrando esta facilidad mediante el uso de la etiqueta *<ISINDEX>* y el procesamiento de *forms* a través de ejemplos como envío de mensajes, interfaz con comandos del sistema o una simple calculadora con las cuatro operaciones básicas.

A la hora de programar CGIs hay que tener presente que se trabaja en un entorno concurrente

Como lo prometido es deuda, en este número de Sólo Programadores se encuentra en el CD-ROM de regalo diversas utilidades para los lectores de esta serie de artículos sobre WEB y lenguaje HTML, desde Browsers, editores de páginas WEB hasta diversidad de informacin técnica y FAQs (*Frequently Asked Questions*) sobre WWW y HTML.

En el subdirectorio *BROWSERS\WINDOWS* se encuentran entre otros *Netscape 1.0*, *Winweb browser*, *Cello browser*, además de *Netscape* para Windows NT y Windows 95 y Linux.

- En el subdirectorio *SERVERS* se encuentran servidores de Web con fuentes, e incluso utilidades para recuperar datos de un form.

Desde estas líneas se anima a todos los lectores a prácticar y descubrir todos los secretos de estas herramientas, que si bien hoy en día no son muy conocidas por los usuarios medios, en menos tiempo de los que muchos imaginan se tornaran en indispensables para todos aquellos que vivimos de este mundo o simplemente somos aficionados a él.

BIBLIOGRAFÍA

Todas las referencias del artículo corresponden a otros artículos publicados por el mismo autor en la revista Sólo Programadores. Las referencias relativas a UNIX son de carácter general y aplicables a otros sistemas operativos (permisos, demonios...)

[Echeva-1], núm 14. "Las shells de UNIX"

[Echeva-2], núm 15. "Tecnología del WWW"

[Echeva-3], núm 5. "Primeros movimientos en UNIX (I)

[Echeva-4], núm 18. "El lenguaje HTML (y III)"

[Echeva-5], Núm 6. "Primeros Movimientos en UNIX (II)"

[Echeva-6], núm 16. "El sistema de ficheros UNIX"

La bibliografía fundamental sobre CGI (y prácticamente la única) se encuentra en: *http://www.w3.org/pub/WWW/CGI*

Los grupos de News relativos al tema son:

comp.infosystems.www.authoring.cgi

comp.infosystems.www.authoring.html

comp.infosystems.www.authoring.images

comp.infosystems.www.authoring.misc

comp.infosystems.www.advocacy (Lo mejor y lo peor)

comp.infosystems.www.announce (Moderated)

comp.infosystems.www.misc

CONTACTAR CON EL AUTOR

Fernando J. Echevarrieta

E-mail internet: echeva@dit.upm.es

E-mail Compuserve: 100646,2456

WWW: http://highland.dit.upm.es:8000

ESTRUCTURAS DE DATOS: ÁRBOLES

José C. Remiro

En este artículo se presentan ideas básicas sobre el tratamiento de árboles binarios, además se incluye una descripción sobre los árboles avl (adelson, velskii y landis) adecuados para realizar procesos de ordenación y búsqueda

El presente artículo continua con la descripción de estructuras dinámicas de datos, esta vez dedicado a los árboles, que a diferencia de las estructuras lineales de datos lleva implícita la idea de bifurcación, pues a partir de un nodo puede accederse de forma directa a varios "sucesores". Además, como se describirá a continuación, su definición recursiva hace que los diferentes procedimientos que pueden aplicarse a esta estructura sean fácilmente implementables mediante técnicas recursivas.

Se puede definir una estructura de árbol, con un tipo base de la siguiente forma: un árbol o bien es una estructura vacía, o bien posee un nodo que apunta a una serie de estructuras árbol disjuntas entre sí, denominadas subárboles.

Los conceptos que se utilizarán en el siguiente artículo serán los siguientes:

- Al nodo superior de un árbol se le denominará **raíz**. Este no forma parte de ningún subárbol del árbol.
- Se denominará **rama** a cada uno de los subárboles que posee un nodo del árbol.
- La **profundidad** o **altura** de un árbol será el número de niveles que posee el árbol. A la raíz se le asignará profundidad 1.
- A los nodos que no disponen de ningún subárbol se les denominará elemento terminal u **hoja**, mientras que a un nodo que posea algún subárbol se le denominará **nodo interior**.
- El **grado** de un árbol será el número máximo de subárboles que puede tener un nodo.

En el cuadro 1 pueden verse indicadas cada una de las definiciones dadas anteriormente. La representación seleccionada muestra la raíz en lo alto de la estructura, mientras que las hojas se disponen en la parte inferior de la estructura.

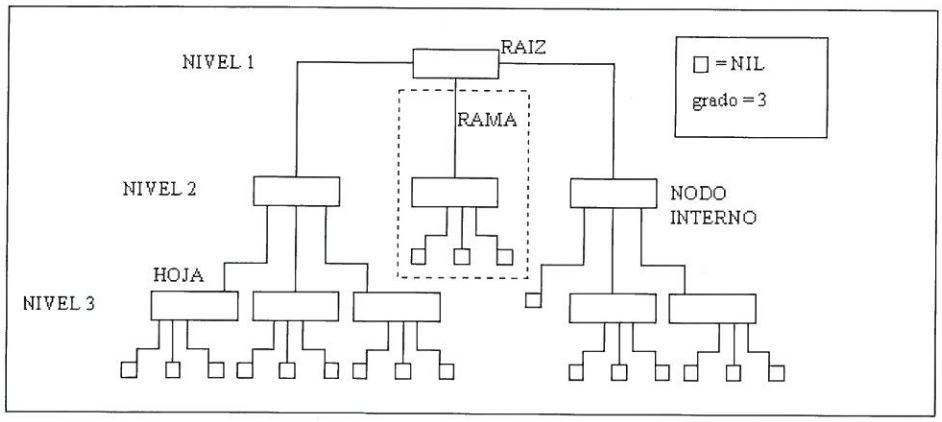
ARBOLES BINARIOS.

De especial importancia como estructura de datos se encuentran los árboles binarios. Estos son árboles de grado 2 (un nodo puede tener a lo sumo dos subárboles) que o bien se encuentra vacío o bien dispone de dos subárboles a los que se denominarán derecho e izquierdo.

Un ejemplo típico de árbol binario lo constituye la representación de las eliminatorias de un campeonato de fútbol. En cada nodo del árbol se encontraría el ganador del enfrentamiento entre los equipos cuyos nodos son apuntados por dicho nodo.

Para la implementación de este tipo de datos se utilizará un nodo que contendrá la información a tratar y dos punteros que señalarán a los subárboles derecho e izquierdo. Si el lenguaje en el que se desea implementar la estructura no dispone de memoria dinámica, se puede utilizar una estructura de tipo array cuyas componentes sean de tipo registro, que alberguen la información a tratar y dos campos de tipo entero que señalen a las posiciones del array que correspondan a los nodos de los subárboles derecho e izquierdo, evidentemente la gestión de los nodos (obtención y liberación de nodos) es responsabilidad del programador. Esta técnica es similar a la desarrollada en artículos anteriores por lo que no se desarrollará en éste.

En el cuadro 2 se muestra un algoritmo recursivo, para crear un árbol de mínima altura, que contenga a los elementos de tipo real contenidos en un



Cuadro 1. Representación de elementos de un árbol.

array de 100 elementos. Este árbol se encuentra equilibrado, en el sentido en que el número de nodos del subárbol izquierdo difiere a lo más en una unidad del número de nodos del subárbol derecho. En el cuadro 3, se muestra de manera gráfica el concepto de árbol equilibrado. El procedimiento **nodo** es el que realiza todo el trabajo, calculando previamente el número de nodos que deberá contener los subárboles derecho e izquierdo (la mitad del número pasado como parámetro), insertando en el nodo actual el número que se encuentra en la posición indicada por la variable **índice**. La ventaja reportada por un árbol balanceado es evidente, ya que se aprovechan al máximo los punteros de los nodos que apuntan a los subárboles. Además, si el proceso de búsqueda dentro de un árbol es importante, cada vez que sea preciso decidir si un elemento se encuentra en alguno de los subárboles quedará reducido a la mitad el número de elementos donde buscar. Este tema será tratado posteriormente.

Es posible diseñar un algoritmo de tipo iterativo para el anterior algoritmo, empleando una estructura de tipo pila que almacene los datos del subárbol que no serán tratados de forma inmediata, de tal forma que una vez construido en profundidad uno de los subárboles, se pueda desapilar la información para construir el resto del árbol (esta idea ya fue desarrollada en el artículo dedicado a recursividad).

Para realizar operaciones sobre un árbol es conveniente dotar a éste de un orden con el fin de simplificar las operaciones a aplicar sobre él. Existen tres formas diferentes de asociar un orden a un árbol, dependiendo de qué elemento del árbol se considere primero. Así, se tratará un árbol en pre-orden cuando el primer elemento del árbol sea la raíz y posteriormente, y en este orden, los subárboles izquierdo y derecho. Se asgnará un orden central, cuando primero se considere el subárbol izquierdo, posteriormente la raíz y por último el subárbol derecho. El último orden que puede asignarse a un árbol es post-orden, cuando primero se considere el subárbol izquierdo, a continuación el subárbol derecho y por último la raíz.

En el cuadro 4, se presentan 3 algoritmos recursivos para recorrer un árbol en los tres órdenes anteriores. Como ejemplo más elaborado, en el cuadro 5 se describe un algoritmo para imprimir un árbol teniendo en cuenta la profundidad de éste, con una indentación para cada uno de sus niveles. Este algoritmo recorre el árbol en orden central.

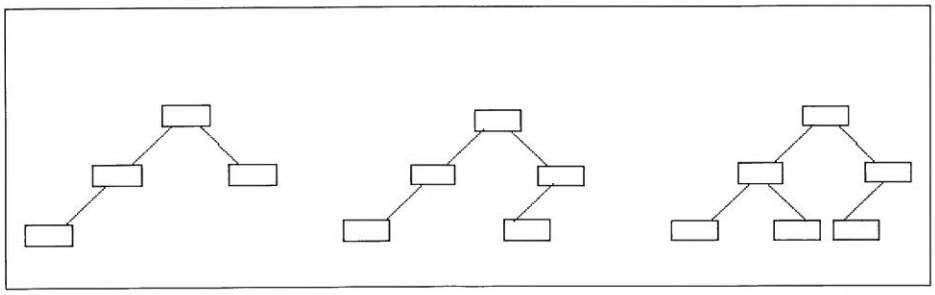
Una vez especificados los diferentes órdenes que pueden establecerse en un

```
procedimiento construye_arbol(secuencia: array[100] real)
 indice: entero
arbol = ↑nodo_arbol
nodo_arbol = registro
                inf: real
                izd, dcho: ↑nodo_arbol

 procedimiento nodo (n: entero; a: ↑nodo_arbol)
  aux: ↑nodo_arbol
  num_izd, num_dcha: entero
 inicio
  Si n = 0
   entonces
     a = nil
   en otro caso
     num_izd = n / 2
     num_dcha = n -num_izd - 1
     obtener_mem (aux)
     indice = indice + 1
     aux↑.inf = secuencia [indice]
     nodo (num_izd, aux↑.izd)
     nodo (num_dcha, aux↑.dcho)
   Fin Si
   a = aux
 fin
inicio /* del procedimiento construye_arbol*/
 indice = 0
 nodo (100, arbol)
fin
```

Cuadro 2. Algoritmo para la creación de un árbol.

árbol, se va a proceder a describir las operaciones básicas en su tratamiento: la búsqueda, la inserción y la eliminación de un nodo. En el cuadro 6 se encuentra un algoritmo para crear un árbol de búsqueda con inserción. En principio, se considera que el árbol se encuentra vacío. Tras la ejecución, el árbol contendrá las palabras que se encuentran en un archivo de texto junto con el número de veces que aparecen. Cada vez que se obtiene una palabra se busca el nodo que debería ocupar, si ésta no se encuentra, habrá que insertar un nuevo nodo que contendrá la nueva palabra junto con su contador inicializado a uno. Si la palabra ya se encuentra en el árbol, se incrementa en uno su contador.

Es claro que se debe establecer un orden dentro del árbol, de tal forma que se pueda decidir donde se encuentra, o debería encontrarse, el nodo correspondiente. El orden utilizado en este algoritmo es el orden central, de tal forma que las palabras que tienen un orden menor que la palabra contenida en un nodo se situarán en el subárbol izquierdo, las palabras de mayor orden lexicográfico que la del correspondiente nodo en su subárbol derecho. Por tanto, si se desea obtener todas las palabras del texto junto con el número de veces que aparecen, en orden lexicográfico, habrá que recorrer el árbol en orden central. A partir de este algoritmo, se pueden diseñar algoritmos de ordenación y búsqueda que sustituyan a los clásicos algoritmos basados en arrays.



Cuadro 3. Árboles perfectamente equilibrados con 4, 5 y 6 nodos.

```
procedimiento preorden (arbol: ↑puntero_arbol)
 inicio
  Si arbol <> nil
    entonces
      Escribir (arbol↑.inf)
      preorden (arbol↑.izd)
      preorden (arbol↑.dcho)
  Fin Si
 Fin

procedimiento orden_central (arbol: ↑puntero_arbol)
 inicio
  Si arbol <> nil
    entonces
      orden_central(arbol↑.izd)
      Escribir (arbol↑.inf)
      orden_central (arbol↑.dcho)
  Fin Si
 fin

procedimiento postorden (arbol: ↑puntero_arbol)
 inicio
  Si arbol <> nil
    entonces
      postorden (arbol↑.izd)
      postorden (arbor↑.dcho)
      Escribir (arbol↑.inf)
  Fin Si
 fin
```

Cuadro 4. Algoritmos para recorrer un árbol en pre-orden, orden central y post-orden.

El cuadro 7 muestra un algoritmo para eliminar un nodo del árbol que contiene una palabra pasada como parámetro. El algoritmo no es tan sencillo como el anterior, pues hay que preservar el orden anteriormente establecido, para la descripción se distinguirán varios casos. Tras localizar el nodo a eliminar, si éste es un nodo terminal o bien solamente tiene un único descendiente el caso es sencillo pues basta con sustituir el nodo a eliminar con el subárbol, si es este el caso, o bien eliminar directamente el nodo si éste es terminal. Si por el contrario el nodo posee los dos subárboles, habrá que descender por el subárbol izquierdo a lo largo de la rama derecha, sustituyendo la información del nodo a borrar por la de dicho nodo. Una vez trasladada la información se procede a la eliminación del nodo del que se trasladó la información. Esta descripción se corresponde con la idea de buscar el mayor de los elementos que se encuentra en el subárbol izquierdo, situándolo en el nodo a eliminar, para así mantener en el subárbol izquierdo del nodo a eliminar todos los elementos menores que la información contenida en dicho nodo. El proceso anteriormente descrito lo realiza el procedimiento recursivo *max_min*.

ARBOLES AVL

Se había comentado anteriormente el concepto de árbol balanceado. Estos árboles presentan la mitad de sus nodos en el subárbol izquierdo y la otra mitad en el subárbol derecho. Si se conoce a priori el número de nodos a insertar en él es fácil diseñar un algoritmo para construir este tipo de árboles, pero si además se desea construir un árbol balanceado ordenado que sopor-

```
procedimiento visualiza_arbol (arbol: ↑nodo_arbol; num_tab: entero)
 contador: entero
inicio
 Si arbol <> nil
   entonces
     visualiza_arbol (arbol↑.izd, num_tab + 1)
     Desde contador = 1 hasta num_tab
           escribir (" ")
           escribir (arbol↑.inf)
           visualiza_arbol ( arbol↑.dcho, num_tab + 1)
      fin desde
fin
```

Cuadro 5. Algoritmo para visualizar un árbol.

te las operaciones de inserción y eliminación el esfuerzo para mantener el árbol balanceado puede ser muy alto. Por otra parte, si se permite que el árbol no esté balanceado, puede ocurrir que al construirlo degenere en una simple lista lineal, con la consiguiente pérdida en la eficacia de la operación de búsqueda (pensemos que tras una comparación en un árbol balanceado la mitad de los nodos del árbol son desechados, mientras que la búsqueda en una lista lineal es especialmente ineficiente).

Los árboles /AVL/, tratan de generar árboles ordenados cuya estructura se asemeje a la de un árbol balanceado, pero de tal forma que las operaciones para reequilibrarlo, tras una operación de inserción o eliminación, sean sencillas.

El criterio de equilibrio para los árboles AVL es el siguiente: "Un árbol está equilibrado siempre que las alturas de los subárboles de todos sus nodos difieran a lo más en una unidad".

Este requisito es menos restrictivo que el correspondiente a un árbol balanceado, de todas formas todos los árboles balanceados son árboles AVL, pero estos últimos mantienen una estructura más que razonable permitiendo además realizar operaciones relativamente sencillas sobre ellos sin que por ello pierdan el equilibrio AVL (cualquier referencia de equilibrio que se realice a continuación se referirá a equilibrio en el sentido de árboles AVL).

Si se tiene en cuenta la inserción de un nuevo nodo en un árbol AVL de tal forma que aumente una unidad la altura del árbol en su subárbol derecho, pueden ocurrir tres casos. Si ambos subárboles tienen igual altura, el árbol tras la inserción seguirá estando en equilibrio, pues las alturas solamente se diferenciarán en una unidad. Si la altura del subárbol izquierdo es una unidad mayor que la del subárbol derecho no se perdería el equilibrio pues ambas alturas se igualarían tras la inserción. Por último, si la altura del subárbol derecho es una unidad mayor que la del subárbol izquierdo, entonces se romperá el equilibrio tras la inserción.

Teniendo en cuenta el cuadro 8, la inserción de los elementos 15, 20, 30 y 40 producirían un desequilibrio en el árbol 1. Por otra parte, la inserción de los elementos 18 o 25 producirían un desequilibrio en el árbol 2. Ambos ejemplos reflejan los únicos casos, salvo casos simétricos, en la pérdida de equilibrio tras una inserción en un árbol AVL.

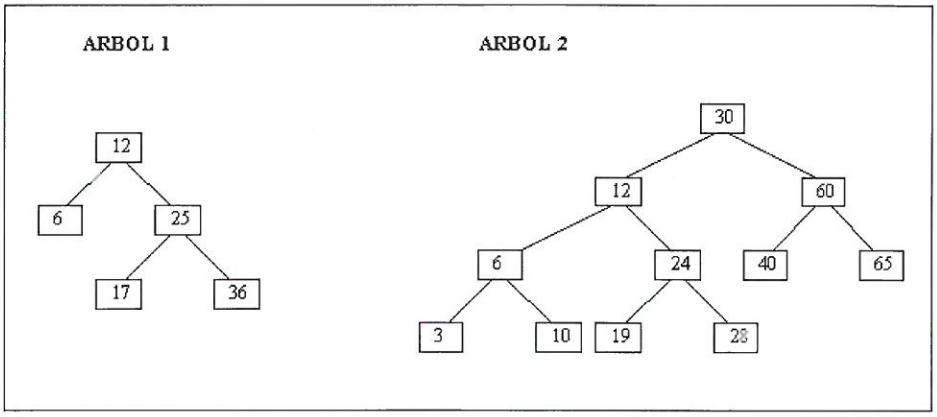
La forma en que un algoritmo compruebe si se ha producido un desequilibrio y su tipo puede realizarse de dos maneras diferentes. Una de ellas consiste en inspeccionar el árbol cada vez que se produce una inserción o una eliminación, o bien asignar información extra a los nodos del árbol contenida en un campo de tipo entero y cuyos valores estén restringidos a 0, 1 y -1. Este valor se obtendrá calculando para cada nodo la diferencia entre las alturas de los subárboles izquierdo y derecho,este diseño es el elegido para la implementación de los algoritmos, de tal forma que al tipo de nodo

```
procedimiento crea_arbol
 arch: Archivo_Texto
 palabra: cadena [20] caracter
 raiz : ↑nodo_arbol
 nodo_arbol registro
              izd, dcho: ↑ nodo_arbol
              inf: cadena [20] caracter
              contador: entero
procedimiento busca_inserta (p: cadena[20] caracter; var aux_arb: nodo_arbol)
 inicio
  Si aux_arb = nil
    entonces /* la palabra no se encontraba en el árbol. Se inserta */
      obtener_mem (aux_arb)
      aux_arb↑.inf = p
      aux_arb↑. contador = 1
      aux_arb↑. izd = nil
      aux_arb↑.dcho = nil
    en otro caso
      Si p < aux_arb↑.inf /*la palabra de estar, se encuentra en el subárbol izquierdo*/
        entonces
          busca_inserta (p, aux_arb↑.izd)
        en otro caso
          Si p > aux_arb↑. inf /*la palabra de estar, se encuentra en el subárbol derecho */
              entonces
                busca_inserta (p, aux_arb↑.dcho)
              en otro caso /*la palabra ya se encontraba en el árbol. Se incrementa en 1 el contador de
              ocurrencias para la palabra*/
                aux_arb↑.contador = aux_arb↑.contador + 1
              Fin si
          Fin si
   Fin si
 Fin
inicio /* de crea_arbol*/
 raiz = nil
 Asociar (arch,"archivo.txt")
 Abrir (E, arch)
 Obtener_palabra (palabra)
 Mientras no EOF (arch)
       busca_inserta (palabra, raiz)
       Obtener_palabra (palabra)
 Fin mientras
fin
```

Cuadro 6. Algoritmo para crear un árbol de búsqueda con inserción.

```
procedimiento eliminar (p: cadena[20] caracter; var arbol:↑ nodo_arbol)
  aux:↑ nodo_arbol
  procedimiento max_min (var ar:↑nodo_arbol)
    inicio
      Si ar↑.dcho <> nil
        entonces
          max_min ( ar↑.dcho)
        en otro caso
          aux↑.inf = ar↑.inf
          aux↑.contador = ar↑.inf
          aux = ar
          ar = ar↑.izd
      fin si
    fin
  inicio
  Si arbol = nil
    entonces
      escribir ("palabra no encontrada")
    en otro caso
      Si p < arbol↑.inf
        entonces /*continua la búsqueda*/
          eliminar (p, arbol↑.izd)
        en otro caso
          Si p > arbol↑.inf /*continua la búsqueda*/
            entonces
              eliminar (p, arbol↑.dcho)
            en otro caso
              aux = arbol
              Si aux↑.dcho = nil /*El nodo a eliminar solo tiene subárbol izquierdo o es terminal*/
                entonces
                  arbol = aux↑.izd
                en otro caso
                  Si aux↑.izd = nil /*El nodo a eliminar solo tiene subárbol derecho*/
                    arbol = aux↑.dcho
                  en otro caso /*el nodo a eliminar tiene ambos subárbol derecho e izquierdo*/
                    max_min (aux↑.izd)
fin si
fin
```

Cuadro 7. Algoritmo para eliminar un nodo.

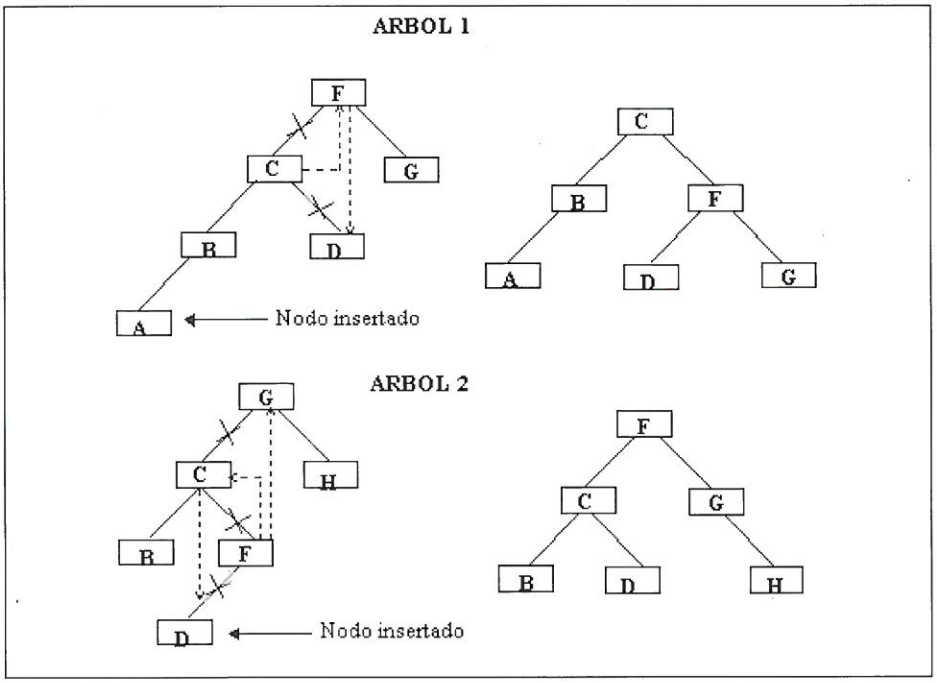


Cuadro 8. Ejemplos de pérdida de equilibrio AVL.

del árbol le ha sido añadido un nuevo campo, *factor*, que contendrá uno de los valores anteriores según corresponda. Así, si la información contenida en el campo es -1, entonces la altura del subárbol derecho del nodo es una unidad mayor que la altura del subárbol izquierdo.

En el cuadro 9 puede observarse de forma gráfica el tipo de inserciones que pueden desequilibrar un árbol y como se resolverían (los nuevos enlaces a crear están indicados con líneas discontinuas, mientras que los enlaces que se pierden están tachados).

El algoritmo para realizar una inserción en un árbol AVL, se basa en el algoritmo para árboles no balanceados, comenzando con la inserción del nodo en el lugar que le correspondería de no estar balanceado. Se observará que hay un parámetro de tipo lógico adicional, dedicado a indicar, tras la inserción, si se ha producido un aumento de la altura en el correspondiente subárbol, si no ha ocurrido esto el árbol seguirá balanceado. Por el contrario, si se ha producido un aumento en la altura, cuando una llamada recursiva devuelve el control a la llamada que lo invocó se comprueba si se ha producido un desequilibrio a partir del nodo que quedó por procesar. Supuesto que el desequilibrio es consecuencia de un aumento de altura debido a la rama izquierda (de forma simétrica sería tratado con la rama derecha) y el factor de equilibrio del nodo a tratar es 0, el factor tras la inserción pasará a ser -1 no habiendo desequilibrio. Si el factor de equilibrio del nodo es 1, el factor pasa a ser 0, no habiendo desequilibrio. Por último, si el factor de equilibrio es -1, entonces se ha producido un desequilibrio, en este caso será necesario reubicar los nodos siguiendo la idea expresada en el cuadro 9 y evidentemente inicializar los factores de equilibrio de cada uno de los nodos implicados en estos movimientos. El anterior proceso se realiza en las instrucciones *en caso de*. Obsérvese la simetría que presentan los casos en que la inserción se ha realizado en el subárbol derecho y en el izquierdo, y como la condición para establecer si se está en uno de los dos casos posibles de pérdida de equilibrio pasa por comprobar que valor presenta el factor de equilibrio de la rama afectada por la inserción.



Cuadro 9. Inserción con desequilibrio y solución.

La eliminación de un nodo en un árbol AVL es, al igual que en el caso de árboles no balanceados, más complicada que la inserción de un nodo. Lo más sencillo es la eliminación de un nodo terminal o bien aquel que posee un único subárbol. Las operaciones a realizar en la eliminación de un nodo son básicamente las mismas que en el caso de la inserción, pero los movimientos de los enlaces de los nodos serán más complicados pudiéndose extender a todos los nodos de una misma rama. Si el nodo a eliminar tiene dos subárboles, entonces el nodo eliminado será sustituido por el nodo más a la derecha de su subárbol izquierdo, es decir, de los nodos que contienen una información menor que el nodo a borrar, el mayor.

LOS LIBROS DE INFORMÁTICA MÁS ACTUALES AL MEJOR PRECIO

WINDOWS 95

2ª EDICIÓN

Un nuevo concepto de la informática

Contiene CD-ROM con una selección de ficheros de sonido, vídeo e imágenes, así como las utilidades disponibles actualmente para Windows 95.

El CD-ROM contiene dibujos, imágenes, animaciones, librerías de bloques y demostraciones de AutoDesk, que podrán ser visualizadas con las utilidades incluídas en el propio CD.

EXPLORANDO INTERNET

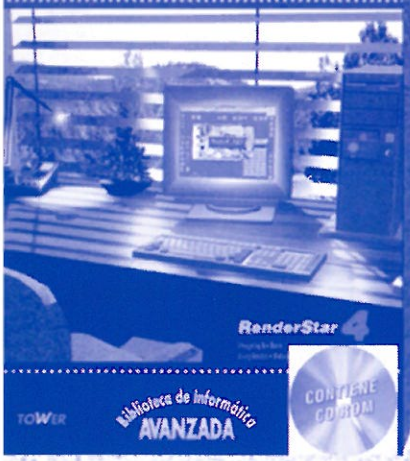
las Autopistas de la Información

Incluye CD-ROM con 800 programas de acceso rápido a INTERNET

Contiene CD-ROM con una demostración de 3D Studio V.4, ficheros de animación, archivos de imágenes ficheros de objetos, demostraciones de rutina IPAS y utilidades para el tratamiento y la visualización de imágenes y animaciones.

AUTOCAD 13

Para dominar el diseño técnico



CADA LIBRO POR SÓLO:

1.995 ptas.
IVA INCLUIDO

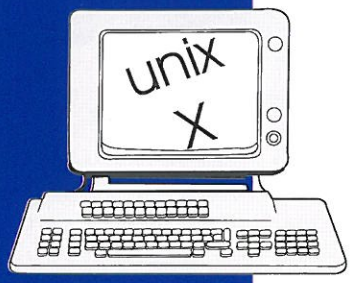
TOWER COMMUNICATIONS

Solicite catálogo al Tel. (91) 661 42 11

3D STUDIO V.4

Las tres dimensiones a su alcance

SUM RPC: LLAMADAS A PROCEDIMIENTO REMOTO (II)



Fernando J. Echevarrieta

Una vez expuestos los conceptos necesarios para la programación con RPCs de SUN, se completa este mes el tema con el desarrollo de una aplicación distribuida

El pasado mes se presentaron los conceptos básicos de programación mediante RPCs, partiendo de su direccionamiento y registro a través del demonio *portmapper* y pasando por las consideraciones necesarias acerca de conversión de semántica y paquetes de usuario. Al final se presentaba una sencilla aplicación y se indicaba el punto de partida para su transformación en una aplicación distribuida mediante un diseño orientado a la aplicación basado en RPCs.

En el presente artículo se retoma aquel punto de partida para el desarrollo paso a paso de la aplicación. Como se puede apreciar en la figura 1, este tipo de desarrollo involucra un alto número de ficheros y relaciones entre los mismos. Sin embargo, no presenta ninguna dificultad conceptual más allá de las expuestas en el artículo del mes anterior, debido a que la secuencia de pasos a seguir es la misma en todos los casos. Aunque el ejemplo a desarrollar le servirá de guía para cualquier desarrollo futuro, sería conveniente que diera un repaso al artículo del mes pasado con el fin de asentar conceptos.

El desarrollo de la aplicación comenzaba con el de una aplicación local que solicitaba un número de tarjeta de crédito mediante el procedimiento *pidedatos* y confirmaba su validez mediante otro llamado *confirmadatos*. El primer paso realizado consitió en dividirla en dos partes. La parte que se convertirá en cliente contenía el *main* y *pidedatos* y la que se convertirá en servidora contenía a *confirmadatos*. Esta última se alojaría en la máquina remota en la que se encontraría el fichero de consulta. El siguiente paso en el desarrollo es la generación de una especificación para el programa *rpcgen*, para lo cual es necesario disponer de unas nociones del *lenguaje XDR*.

EL LENGUAJE XDR

Una de las cuestiones más importantes que afectan al mundo de los sistemas abiertos es el de la representación de la información. Es por ello que sobre los cuatro niveles bajos de la comunicación, físico, de enlace, de red y de transporte, es necesario asentar un nivel dedicado a la representación de los datos. En el caso de Internet, con la arquitectura TCP/IP, en la que no existe una división clara de niveles, se suele emplear el lenguaje *XDR* (*eXternal Data Representation*) para comunicar tipos de datos entre máquinas heterogéneas. Imagine el lector que, en el más sencillo de los casos, se podría colocar el cliente que se va a desarrollar en PCs con FreeBSD o Linux y realizar consultas a una máquina servidora de SUN. Si se transmitieran los datos tal cual, surgiría ya un problema al tratar cliente y servidor con un convenio diferente de representación de enteros.

El mecanismo de RPC se sustenta sobre el de sockets transmitiendo la información codificada en XDR por lo que será necesario conocer este lenguaje. En cada máquina habrá una

librería de gestión de XDR que se encargará de realizar la conversión del formato de la máquina a XDR y viceversa. Si se emplea el programa *rpcgen*, no será necesario conocerla ya que este programa genera todo el código de serialización necesario, como se comentaba en el artículo del mes anterior. Sin embargo, sí es necesario indicar a *rpcgen* qué tipos XDR se van a emplear en la comunicación.

XDR no es un lenguaje de programación sino un lenguaje de representación de datos, lo que no debe inducir a confusión. Si este apartado le resulta confuso no se preocupe, irá asentando los conceptos a medida que se avance en el desarrollo de la aplicación.

A continuación se muestra un mínimo conjunto de ejemplos básicos para su empleo en programación con RPCs.

Los tipos XDR pueden ser *simples* o *compuestos* y, aunque en XDR no existen punteros debido a que no existe un mecanismo de direccionamiento global de la memoria de las máquinas en red, sí se permite la definición de tipos recursivos. En la tabla 1 se pueden observar algunos tipos simples.

Para la definición de tipos XDR se utiliza el nombre predefinido *typedef*. Por ejemplo:

```
typedef int numero
```

Es posible definir tipos enumerados mediante la palabra reservada *enum*. En la definición, es obligatorio asignar valores a los componentes. Por ejemplo:

```
enum dias {LUNES=1, MARTES=2, ...}
```

Existen tres tipos de cadenas:

a) Tamaño fijo: Se indica la longitud entre corchetes:

```
typedef int vector[8]
```

b) Tamaño acotado: Se indica la longitud máxima entre los signos "menor que" y "mayor que":

```
typedef string nombre<30>
```

c) Tamaño libre: Se definen igual que un tipo acotado sin indicar límite:

```
typedef string cadena<>
```

La definición de constantes se realiza mediante la palabra reservada *const*:

```
const MAX = 30
```

En la definición de tipos compuestos existe una peligrosa similitud con el C. Así, es posible definir estructuras:

```
struct usuario {
  string nombre<>
  int    edad;
  string *sexo;
}
```

Como se ha comentado, XDR no admite punteros. Así pues, en XDR el asterisco significa que ese campo de la estructura es opcional. Es decir, "en la estructura *usuario* podría aparecer o no un campo *sexo* y si lo hace debe ser de tipo *string*".

Para la representación de información entre máquinas heterogéneas se empleará el lenguaje XDR

También es posible definir uniones que realmente se parecen más a los registros con partes variante de Modula-2 que a las uniones de C. Por ejemplo:

```
union resultado switch (dia d) {
  case LUNES: numero clase;
  case DOMINGO: string cine<>;
  default void;
}
```

ESPECIFICACIÓN RPCGEN

Una vez se disponen de unas nociones de XDR se puede proceder a realizar la especificación *rpcgen*. Esta especificación deberá contener las declaraciones de constantes, tipo y programas remotos, especificando los procedimientos contenidos en cada versión y los tipos de sus parámetros y valores de retorno. Como se puede observar en el ejemplo, un procedimiento remoto sólo admite un único parámetro, por lo que si el procedimiento local original admitía varios, será necesario englobarlos dentro de una estructura (futuras versiones admitirán el paso de múltiples parámetros). Todo ello debe realizarse en lenguaje de programación de RPC, no en C, en el que los tipos se especifican según la sintaxis de XDR.

TABLA 1

unsigned (int)	32 bits
bool	32 bits
unsigned (hyper)	64 bits
float	32 bits
double	64 bits
opaque	no convertido
void	0 (no datos)

Tipos simples XDR.

El lenguaje de RPC es muy similar al C, lo que supone una ventaja y el ejemplo, implementado en el listado 1, muestra un esqueleto genérico. La desventaja surge de que al ser tal la similitud, es muy fácil equivocarse.

Primeramente, se realiza la definición de los datos en XDR.

En segundo lugar, se declaran los programas remotos mediante:

```
program <NOMBRE> {
```

LISTADO 1

```
/* rpccredit.x - Declaraciones en lenguaje de RPC
*/
/* echeva@dit.upm.es para Solo Programadores
*/

const MAXNOMBRE = 30;  /* Definición de una
constante */

typedef string tiponombre<MAXNOMBRE>; /*
Definición de un tipo: tiponombre */

struct datos {  /* Definición de una estructura */
  tiponombre nombre;
  unsigned   numero;
};

program RPCCREDITO {  /* Nombre del progra-
ma remoto */
    version RPCCREDITVERS {    /* Version */
   void    NADA       (void) = 0;  /* Procedimiento
nulo  */
    unsigned CONFIRMADATOS (datos) = 1;  /*
Procedimiento remoto */
   } = 1;          /* Numero de version    */
} = 0x10000008;     /* Numero de programa   */
```

Especificación rpcgen.

<declaración_prog> } = núm_programa;

Toda declaración de programa remoto debe contener al menos una declaración de versión que se realiza mediante:

version <VERSION> { <declaración_vers> } = núm_version;

En la declaración de la versión figuran las declaraciones de cada procedimiento remoto indicando los tipos XDR del parámetro que acepta y de los valores que devuelve de la forma:

<tipo_retorno> proc_remoto (<tipo_param>) = núm_proc;

De esta forma es como se declara la terna (*prog*, *vers*, *proc*) que define unívocamente un RPC como se explicó en el artículo anterior. Para realizar cualquier aplicación será suficiente tomar el esqueleto del listado 1 e indicar los tipos necesarios mediante lo expuesto en el apartado anterior.

EJECUCIÓN DE RPCGEN

El programa *rpcgen* es una utilidad que, a partir de un fichero de especificaciones en lenguaje de RPCs genera ficheros fuente en C con la mayor parte del código necesario para las partes cliente y servidor. Así, proporciona el código para los cabos cliente y servidor, serialización de argumentos y resultados, envío de RPCs y respuestas; aunque aún será necesario que el programador añada algo de código.

El programa *rpcgen* toma como entrada un fichero con extensión .x como el realizado en el paso anterior y al que se ha llamado *rpccredit.x*. Mediante la ejecución de

rpcgen rpccredit.x

se obtendrán cuatro ficheros con código C como salida: *rpccredit.h*: con la definición de constantes y tipos usados por el cliente y el servidor así como de los procedimientos remotos

rpccredit_xdr.c: conjunto de procedimientos XDR utilizados por el cliente y el servidor

rpccredit_clnt.c: comunicaciones de los cabos clientes *rpccredit_svc.c*: comunicaciones del servidor con código encargado de obtener el puerto del protocolo, registrar el programa RPC en el *portmapper*, esperar llamadas y repartirlas al cabo del servidor apropiado y devolver sus respuestas.

El código contenido en estos ficheros es lo que se considera el bajo nivel del mecanismo de RPC por lo que únicamente tendrá interés para los lectores que deseen profundizar más. El único fichero que puede resultar necesario consultar en algún caso es el .h, ya que contiene las definiciones en C de los datos que se habían expresado en XDR.

En las últimas versiones de rpcgen, si se le invoca mediante

rpcgen -a rpccredit.x

se obtendrán dos ficheros adicionales *rpccredit_server.c* y *rpccredit_client.c* con los esqueletos del código necesario para las interfaces con los cabos servidor y cliente que debe escribir el programador, respectivamente. El significado de esto es que a efectos de implementación resulta más claro dividir los cabos en dos partes, las que

La aplicación ejemplo es una consulta remota de validez de tarjetas de crédito

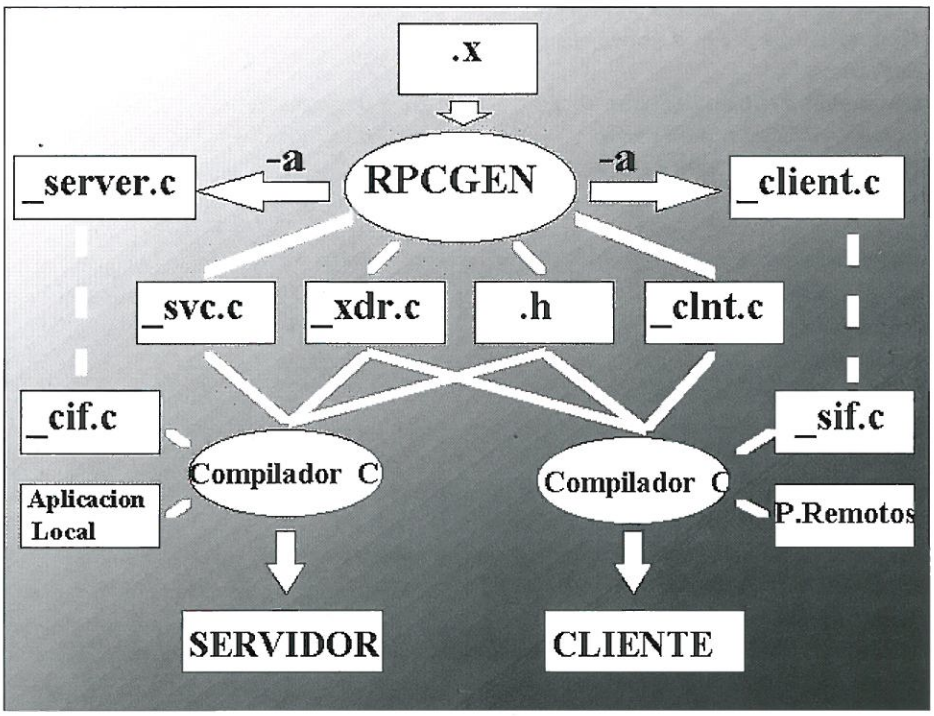


Figura 1. Relaciones entre los ficheros correspondientes a las etapas de la realización de una aplicación distribuída mediante rpcgen.

LISTADO 2

```
/* rpccredit_sif.c - Interfaz con el cabo servidor */
/* echeva@dit.upm.es para Solo Programadores
*/

#include "rpccredit.h"

void *
nada_1_svc(void *argp, struct svc_req *rqstp)
{
        static void *nulo;
        return nulo;

} /* end nulo_1 */

u_int *
confirmadatos_1_svc(datos *argp, struct svc_req
*rqstp)
{
            static u_int  confirmacion;

confirmacion=confirmadatos(argp->nombre,argp-
>numero);
            return &confirmacion;

} /* end confirmadatos_1_svc */
```

Interfaz con el cabo servidor.

LISTADO 3

```
/* rpccredit_cif.c - Interfaz con el cabo cliente */
/* echeva@dit.upm.es para Solo Programadores
*/

#include "rpccredit.h"

extern CLIENT *clnt;

confirmadatos(char *nombre, int numero)
{
  u_int  *result;
  datos  confirmadatos_1_arg;

  confirmadatos_1_arg.nombre = nombre;
  confirmadatos_1_arg.numero = numero;

  /* Cortado y pegado del fichero _client óóóóó */
   result=confirmadatos_1(&confirmadatos_1_arg,
clnt);
     if (result == NULL) {
        clnt_perror(clnt, "call failed:");
      }
        /* ó6ó6ó6ó6ó6ó6ó6ó6ó6ó6ó6ó6ó6ó- */

           return *result;
}
```

Interfaz con el cabo cliente.

se han denominado *comunicaciones del cabo* y las *interfaces de programa con los cabos*, como muestra la figura 2. Esto se debe a que las interfaces, al tener que coincidir con las de la aplicación, como se indicaba en el punto anterior, se dejan en manos del programador. Sin embargo, la generación del mecanismo de comunicaciones, es la parte más compleja y únicamente depende de lo especificado en el fichero .x.

ESCRITURA DE PROCEDIMIENTOS INTERFAZ DE CABOS

Para la realización de las interfaces de cabos del servidor se ha empleado el fichero *rpccredit_server.c* generado por la opción -a de *rpcgen* debido a que proporciona un esqueleto muy claro del programa indicando dónde se debe insertar el código. Si se compara el listado 2 con el resultado de *rpcgen* se verá que este paso no es difícil.

En versiones anteriores de *rpcgen* que no incorporan la opción -a, habrá que construir los cabos partiendo desde cero para lo que habrá que observar los siguiente puntos:

1. Habrá que definir un procedimiento "interfaz de cabo" por cada procedimiento remoto definido en el fichero .x y la definición del mismo se podrá extraer del fichero .h. Se podrán observar dos diferencias:

a) los procedimientos a declarar tendrán un único argumento en lugar de los dos que se pueden observar en el listado 4 y

b) las funciones no llevarán el sufijo *_svc*.

2. rpcgen pasa punteros a los objetos en lugar de pasar los propios objetos y como retorno hay que devolver su dirección.

3. Es obligatorio declarar como *static* todos los códigos de retorno.

Todos estos puntos pueden observarse en el listado 2, al que se ha denominado *rpccredit_sif.c* (*Server InterFace*) que también servirá como guía a los lectores con *rpcgen* antiguo. El procedimiento *confirmadatos_1_svc* se limita a la traducción de los argumentos desde la forma RPC a la original y llama al procedimiento *confirmadatos* del subprograma *credit2.c*, con lo que no habrá hecho falta cambiar ni las fuentes ni el fichero objeto, proporcionando una transparencia total a la aplicación original como se indicaba al comienzo del artículo. Obsérvese, que el programador podría haber insertado directamente el código del procedimiento remoto en este fichero, perdiendo transparencia pero disminuyendo el número de módulos necesarios. Es debido a esto y a que *rpcgen* no conoce la aplicación original por lo

El segundo paso en la distribución de una aplicación es la fragmentación de la misma

LISTADO 4

```
/* rpccredit.c - Aplicacion local con el main y el
procedimiento A */
/* echeva@dit.upm.es para Solo Programadores
*/

#include "rpccredit.h"

CLIENT *clnt; /* Generado por rpcgen [1] */

void
rpccredito_1( char* host )
{
        char nombre[30];
    int  numero;

  /* Generado por rpcgen [2]ó6ó6ó6ó6*/
        clnt = clnt_create(host, RPCCREDITO,
RPCCREDITVERS, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror(host);
                exit(1);
        }
  /* ó6ó6ó6ó6ó6ó6ó6ó6ó6ó6-*/

        pidedatos(nombre,&numero);
    if (confirmadatos(nombre,numero) == 1)
      printf("CONFIRMADO\n");
    else
      printf("DENEGADO\n");

clnt_destroy( clnt ); /* Generado por rpcgen [4] */
}

pidedatos(char *nombre, int *numero)
/* Procedimiento A: local */
{
    printf("Nombre: "); scanf ("%s",nombre);
    printf("Numero: "); scanf ("%d",numero);
}

main( int argc, char* argv[] )
/* Generado por rpcgen [4] */
{
        char *host;

        if(argc < 2) {
printf("usage: %s server_host\n", argv[0]);
                exit(1);
        }
        host = argv[1];
        rpccredito_1( host );
}
```

Programa principal cliente

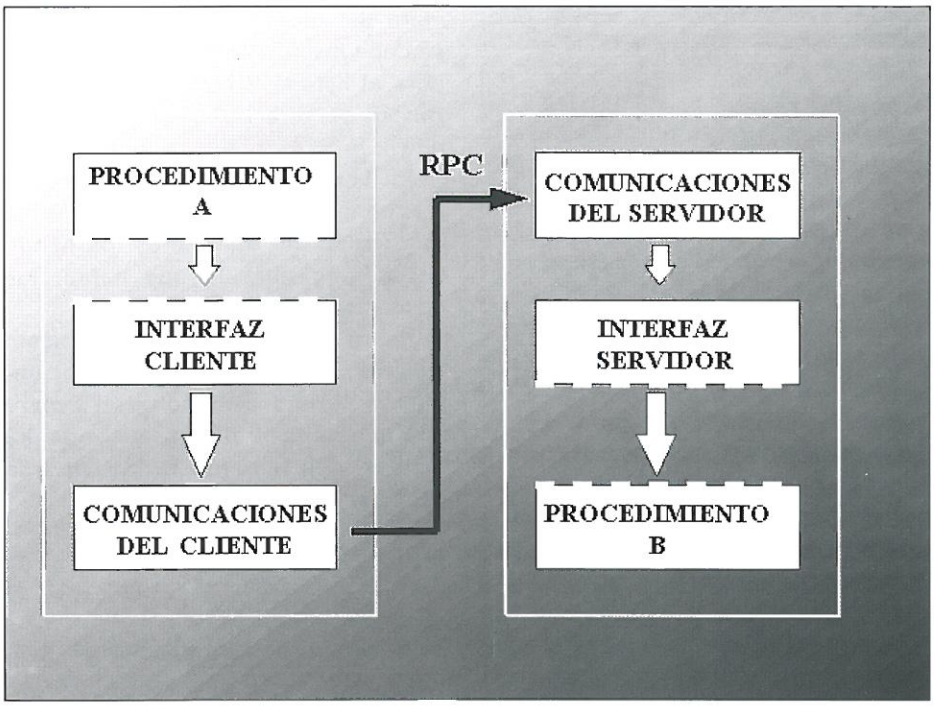
que se hace necesaria la intervención del programador.

El fichero *_client.c* generado por *rpcgen* es, por el contrario, algo menos "elegante" ya que genera un fichero con un procedimiento *main()* y que realiza llamadas "a diestro y siniestro" a todos los procedimientos remotos. Por otra parte "invita" a incluir interfaces de cabos y programa principal cliente en un solo fichero. De todas formas, proporciona un estupendo código para "cortar y pegar" con la garantía de que todo es sintácticamente correcto. En el ejemplo, se ha realizado a mano la división entre el programa principal y los interfaces de cabos, por lo que salvo las líneas indicadas, se ha realizado el listado 3 desde cero, por lo que la situación será la misma sea cual sea la versión de *rpcgen* que esté empleando el lector.

En este caso se ve como en el lado del cliente se definen funciones que presentan el mismo interfaz que los procedimientos originales y que ahora son remotos. Así, el nuevo procedimiento *confirmadatos*, actúa como un impostor para el cliente convirtiendo los argumentos a la forma usada por *rpcgen* y después llama al procedimiento de comunicaciones generado por éste. La principal diferencia en los parámetros es que todos los procedimientos producidos por *rpcgen* utilizan indirección. También será necesario incluir la variable externa *clnt* que se declarará en el programa principal cliente.

MODIFICACIÓN DEL PROGRAMA ORIGINAL

Por último, es necesario realizar algunas modificaciones al programa cliente que contenía al *main()* y a *pidedatos*. En el listado 4 se ha mezclado la máxima cantidad de código posible del fichero *_client.c* generado por las versiones nuevas de *rpcgen* con fines didácticos a costa de perder transparencia y elegancia. De esta forma se puede observar un posible modo de "enganchar" el programa inicial con el nuevo código que crea un *main* propio que toma como argumento el nombre de la máquina en que reside el programa remoto inclu-



División de cabos en módulo de comunicaciones y módulo interfaz de programa.

LISTADO 5

```
# SOLO PROGRAMADORES: APLICACIONES DISTRIBUIDAS EN UNIX CON SUN RPC
#     Autor:  Fernando  J.  Echevarrieta (echeva@dit.upm.es)

# ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-
# Variables
# ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-

CC      = gcc
CFLAGS  = # -g
NOMBRE1 = rpccredit
NOMBRE2 = rpccreditd

all: $(NOMBRE1) $(NOMBRE2)

#ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-
# Generacion de programas mediante rpcgen
#ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-

rpccredit_svc.c : rpccredit.h rpccredit.x

rpccredit_clnt.c: rpccredit.h rpccredit.x

rpccredit_xdr.c : rpccredit.h rpccredit.x

rpccredit.h     : rpccredit.x
                  rpcgen rpccredit.x

#ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-
# Generacion del cliente y el servidor
#ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-ó-

$(NOMBRE1)   : rpccredit.o rpccredit_clnt.o rpccredit_cif.o rpccredit_xdr.o
                $(CC) $(CFLAGS) -o $(NOMBRE1) rpccredit.o rpccredit_clnt.o \
                rpccredit_cif.o rpccredit_xdr.o

$(NOMBRE2)   : credit2.o rpccredit_svc.o rpccredit_sif.o rpccredit_xdr.o
                $(CC) $(CFLAGS) -o $(NOMBRE2) credit2.o rpccredit_svc.o \
                rpccredit_sif.o rpccredit_xdr.o
```

Makefile para la realización de la aplicación distribuída.

yendo un "usage". Obsérvese que la línea [1] ha sido tomada del procedimiento *rpccredito_1* con el objeto de hacerla visible al interfaz realizado en el paso anterior (*rpcgen -a* lo proporciona todo mezclado en el fichero *_client.c*). El consejo del autor es añadir a la mitad cliente original las líneas marcadas con [1] y [2] y prescindir del resto, con lo que estos dos conjuntos de líneas serán los únicos que se habrán realizado al programa original.

En el caso del ejemplo y por no perder generalidad, la máquina remota se indica a través de un argumento al *main*. Sin embargo, en el caso de que el lector fuera un programador de una compañía de tarjetas de crédito, es probable que la máquina remota fuera una constante definiendo la máquina de la compañía. También sería posible, como se adelantó, que el procedimiento *confirmadatos* se presentara como parte de una biblioteca que la compañía facilitara a los programadores de sus clientes y la máquina remota fuera un argumento del mismo o, de nuevo, una constante, con lo que la transparencia sería aún mayor.

Para la ejecución de programas remotos será necesario lanzar el portmapper

COMPILACIÓN, ENLAZADO Y EJECUCIÓN

Como comprobará el lector, el proceso, si bien no es difícil si se sigue paso a paso, resulta bastante complejo, debido a las relaciones de dependencia entre unos y otros ficheros, al número de los mismos y a la repetición de definiciones entre ellos. Por ello, se facilita el *makefile* del listado 5 en el que se definen todas las depencias y operaciones necesarias para la consecución de cliente y servidor. Por último, recuerde que para la utilización de las aplicaciones RPC es necesario que el *portmapper* se encuentre lanzado en su máquina. Para lanzarlo deberá ejecutar el programa *rpc.portmap* que en las distribuciones de Linux se encuentra en el directorio */usr/sbin*.

OTRAS CONSIDERACIONES: CONCLUSIÓN

Los principios aquí expuestos no son más que las bases de la programación con RPC. Como muchos lectores habrán advertido, falta un detalle crucial: la seguridad. El mecanismo de RPC proporciona distintos medios de seguridad y autenticación. Tampoco se ha estudiado la biblioteca de RPC de bajo nivel, ni el control de flujos XDR. Sin embargo, con lo expuesto, el lector dispone ya de uno de los mecanismos más potentes para la generación de aplicaciones distribuidas. Si desea profundizar más no será una mala lectura para comenzar, el texto "*Internetworking with TCP/IP vol. III. Client-Server programming and applications BSD socket version.* Douglas E. Comer & David L. Stevens". Entre tanto, siga con paciencia los pasos descritos y si tiene alguna duda puede contactar con el autor mediante carta a la redacción de la revista o a través:

E-mail internet: echeva@dit.upm.es
E-mail compuserve: 100646,2456
o dejar un mensaje en el servidor WWW:
http://highland.dit.upm.es:8000

OTRAS CONSIDERACIONES: CONCLUSIÓN

Los principios aquí expuestos no son más que las bases de la programación con RPC. Como muchos lectores habrán advertido, falta un detalle crucial: la seguridad. El mecanismo de RPC proporciona distintos medios de seguridad y autenticación. Tampoco se ha estudiado la biblioteca de RPC de bajo nivel, ni el control de flujos XDR. Sin embargo, con lo expuesto, el lector dispone ya de uno de los mecanismos más potentes para la generación de aplicaciones distribuidas. Si desea profundizar más no será una mala lectura para comenzar, el texto "*Internetworking with TCP/IP vol. III. Client-Server programming and applications BSD socket version.* Douglas E. Comer & David L. Stevens". Entre tanto, siga con paciencia los pasos descritos y si tiene alguna duda puede contactar con el autor mediante carta a la redacción de la revista o a través:

E-mail internet: echeva@dit.upm.es
E-mail compuserve: 100646,2456
o dejar un mensaje en el servidor WWW:
http://highland.dit.upm.es:8000

De todas maneras es intención de Sólo Programadores tratar en toda su extensión en próximos números la biblioteca de RPC de bajo nivel así como el control de flujos XDR. Nuestra intención avanzar poco a poco pero con paso firme, para lo cual sería necesario que además de los libros citados arriba, si usted necesita ponerse al día por haberse incorporado tarde a esta serie sobre Unix, o por simplemente creer que el nivel de estas entregas es fuerte, se recomienda la lectura de los libros propuestos en la tabla adjunta, donde se proporciona una serie de títulos básicos y de imprescindible lectura. También se recomienda un repaso a todos los artículos hasta ahora publicados para así además de repasar y asimilar los conceptos estudiados, estar perfectamente preparados para los siguientes puntos a tratar, los cuales son eminentemente prácticos. Temas como la Arquitectura Cliente-Servidor son de una importancia suprema debido a que uso extendido en cientos de aplicaciones hacen imprescindible que el profesional de este sector domine con máxima fluidez

BIBLIOGRAFÍA

- " Unix System V "; Antonio P. Gimenez
  Anaya Multimedia, 1994
- " Unix" ; George W. Leach;
  Addison Wesley, 1994
- " ¡ Aprenda Unix ya ! " ; Jim Gardner
  Anaya Multimedia, 1993
- " Introducción a los Sistemas operativos" A. Tanembaum
  Mc Graw Hill, 1989
- " Unix para impacientes" ; Abrahams y Larson
  Addison Wesley, 1995
- " C manual de referencia "; Herbert Schildt
  Mc Graw Hill, 1995
- " El libro del SCO Unix " ; Joanne Woodcock y Halvorson
  Anaya Multimedia, 1995
- " TCP/ IP illustrated" ; D. Stevens
  Prentice-Hall
- "An introduction to Berkely Unix" ; Paul swang
  Addison Wesley

LISTADO 1

```
{ 3D library                    }
{ Sólo Programadores ver. 1.0   }
{                               }
{       by Crom / Spanish Lords Feb 96 }
Unit Lib3D;
INTERFACE
Const
{ ver. 1.0 works with radians ;'(       }
  RadDeg  = 1*3.1415926535/180;
{ 3D origin    }
  XOrg    : Integer =  0;
  YOrg    : Integer =  0;
  ZOrg    : Integer =  60;
{ Center screen }
  XCenter : Integer = 160;
  YCenter : Integer = 100;
{ Rotations angles              }
  XAngle  : Integer =  0;
  YAngle  : Integer =  0;
  ZAngle  : Integer =  0;

Type
  Point3D = Record
              x : Integer;
              y : Integer;
              z : Integer;
            End;

PROCEDURE Calc3DRotations (Var X,Y,Z: Integer);
PROCEDURE Proyect (X,Y,Z: Integer; Var Xf,Yf: Word);

IMPLEMENTATION
{ Deg2Rad funtion, now we can work with degrees ;}
Function Rad (Deg:Integer):Real;
Begin
  Rad:=Deg*(Pi/360);
End;

Procedure Calc3DRotations (Var X,Y,Z: Integer);
Var
  X1,Y1,Z1 : Integer;
Begin
    X1  :=  Round(cos(rad(YAngle))  *  X  -
sin(rad(YAngle)) * Z);
    Z1  :=  Round(sin(rad(YAngle))  *  X  +
cos(rad(YAngle)) * Z);
    X   :=  Round(cos(rad(ZAngle))  *  X1 +
sin(rad(ZAngle)) * Y);
    Y1  :=  Round(cos(rad(ZAngle))  *  Y  -
sin(rad(ZAngle)) * X1);
    Z   :=  Round(cos(rad(XAngle))  *  Z1 -
sin(rad(XAngle)) * Y1);
    Y   :=  Round(sin(rad(XAngle))  *  Z1 +
cos(rad(XAngle)) * Y1);
End;

Procedure Proyect (X,Y,Z: Integer; Var Xf,Yf:
Word);
Begin
      Xf   :=   XCenter+Trunc((XOrg*Z   -
X*ZOrg)/(Z-ZOrg));
      Yf   :=   YCenter+Trunc((YOrg*Z   -
Y*ZOrg)/(Z-ZOrg));
End;
END.
```

mensionales, para ser transformados en dos dimensiones, es el de transformación bilineal. Pero no entraremos en profundidad en el tema, con el fin de llegar al máximo número de lectores posibles. No obstante serán necesarias una nociones mínimas de álgebra matricial, para comprender el desarrollo de las fórmulas. Aquellos lectores que no posean conocimientos matemáticos suficientes, pueden limitarse a trabajar con las fórmulas finales.

Definiremos tres ángulos, para los giros sobre los ejes. Ángulo de rotación sobre el eje x o ángulo de caída, ángulo de rotación sobre el eje y o ángulo de deriva y ángulo de rotación sobre el eje z o ángulo de balanza. Si se va a usar más de un objeto, es interesante definir ángulos de rotación diferentes, para cada objeto.

Los giros sobre cada eje, producen tres matrices de transformación, una, para cada eje.

Como se puede observar, desarrollar estas matrices da lugar a ecuaciones lineales. También es posible combinar los tres posibles giros para obtener una matriz de transformación general, así como sus ecuaciones lineales:

Una vez transformados los puntos en el espacio, es decir, sometidos a las posibles rotaciones que pueden sufrir, se procede a proyectar los puntos sobre la pantalla del ordenador, teniendo en cuenta la posición del observador dentro del espacio tridimensional, para lo cual, como es de suponer, es necesario conocer, al menos, el punto donde se encuentra dicho observador. En nuestras fórmulas de proyección el punto donde se encuentra será (*XOrg, YOrg, ZOrg*).

Se observa en estas fórmulas la aparición de dos nuevos parámetros, *XScreen* e *YScreen*, estos representan las coordenadas x,y de la pantalla sobre las que se desea centrar nuestro espacio virtual. El lector puede observar que modificar dichos parámetros, produce un "falso" movimiento de traslación dentro de nuestro espacio virtual.

Además se debe destacar un posible problema, cuando la coordenada Z del punto a rotar sea igual a la coordenada z del observador, el divisor de

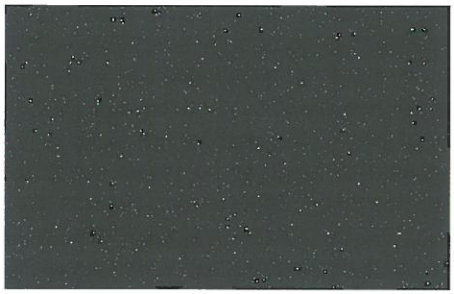
LISTADO 2

```
{ 3D library                    }
{ Sólo Programadores ver. 2.0   }
{                               }
{ Something about this library: }
{ * 360 degrees = 1024 steps    }
{ * SinData are stored in 8 fixed point ;) }
{                               }
{       by Crom / Spanish Lords Feb 96 }
Unit Lib3D_v2;

INTERFACE
Const
{ 3D origin                     }
  XOrg     : Integer =  0;
  YOrg     : Integer =  0;
  ZOrg     : Integer =  60;
{ Center screen                 }
  XCenter  : Integer = 160;
  YCenter  : Integer = 100;
{ Number of points in CosTable  }
  NumSinVal = 256*4;

Type
  Point3D   = Array [1..3] of Integer;

Var
  CosTable  : Array [0..NumSinVal] of Integer;

FUNCTION FastSin  (Angle3D:Integer):Integer;
FUNCTION FastCos  (Angle3D:Integer):Integer;
PROCEDURE Calc3DRotations
  (SinX,CosX,SinY,CosY,SinZ,CosZ:Integer;
   OrgPoints : Array of Point3D;
   Var DesPoints : Array of Point3D;
   NumPoints : Word    );
PROCEDURE Proyect
  (X,Y,Z: Integer;
   Var Xf,Yf: Word;
   XScreen,YScreen:Integer);

IMPLEMENTATION
Procedure MakeCosTable;
Var
  CntVal : Word;
  CntAng : Real;
  IncDeg : Real;
Begin
  IncDeg := 2*PI/NumSinVal;
  CntAng := IncDeg;
  CntVal := 0;
  Repeat
    CosTable [CntVal] := Round(255*cos(CntAng));
    CntAng := CntAng+IncDeg;
    Inc (CntVal);
  Until CntVal>1024;
End;

Function                                FastSin
(Angle3D:Integer):Integer;Assembler;
Asm
  mov  ax,Seg CosTable
  mov  es,ax
  mov  di,Offset CosTable
  mov  dx,Angle3D
  mov  bx,1024
  add  dx,256
```

```
        cmp  dx,bx
  jle  @@Ok
  sub  dx,1024
@@Ok:
  shl  dx,1
  add  di,dx
  mov  ax,es:[di]
End;

Function                                    FastCos
(Angle3D:Integer):Integer;assembler;
Asm
  mov  ax,Seg CosTable
  mov  es,ax
  mov  di,Offset CosTable
  mov  dx,Angle3D
  shl  dx,1
  add  di,dx
  mov  ax,es:[di]
End;

Procedure Calc3DRotations
   (SinX,CosX,SinY,CosY,SinZ,CosZ:Integer;
    OrgPoints : Array of Point3D;
    Var DesPoints : Array of Point3D;
    NumPoints : Word     );
Var
  X1,Y1,Z1  : LongInt;
  Var32_1   : LongInt;
  Var32_2   : LongInt;
  CntPoints : Word;
Begin
  For CntPoints:=0 to NumPoints-1 do
    Begin
{    X1 := (cos(YAngle) * X  - sin(YAngle) * Z);}
     Var32_1 := (CosY*OrgPoints [CntPoints,1]) div
256;
     Var32_2 := (SinY*OrgPoints [CntPoints,3]) div
256;
     X1      := Var32_1-Var32_2;
{    Z1 := (sin(YAngle) * X  + cos(YAngle) * Z);}
     Var32_1 := (SinY*OrgPoints [CntPoints,1]) div
256;
     Var32_2 := (CosY*OrgPoints [CntPoints,3]) div
256;
     Z1      := Var32_1+Var32_2;
{    X  := (cos(ZAngle) * X1 + sin(ZAngle) * Y);}
     Var32_1 := (CosZ*X1) div 256;
     Var32_2 := (SinZ*OrgPoints [CntPoints,2]) div
256;
    DesPoints  [CntPoints,1] := Var32_1+Var32_2;
{    Y1 := (cos(ZAngle) * Y  - sin(ZAngle) * X1);}
     Var32_1 := (CosZ*OrgPoints [CntPoints,2]) div
256;
     Var32_2 := (SinZ*X1) div 256;
     Y1      := Var32_1-Var32_2;
{    Z  := (cos(XAngle) * Z1 - sin(XAngle) * Y1);}
     Var32_1 := (CosX*Z1) div 256;
     Var32_2 := (SinX*Y1) div 256;
    DesPoints  [CntPoints,3] := Var32_1-Var32_2;
{    Y  := (sin(XAngle)) * Z1 + cos(XAngle) * Y1);}
     Var32_1 := (SinX*Z1) div 256;
     Var32_2 := (CosX*Y1) div 256;
    DesPoints  [CntPoints,2] := Var32_1+Var32_2;
   End;
End;

Procedure Proyect (X,Y,Z: Integer; Var
```

```
      Xf,Yf: Word; XScreen,YScreen:Integer);
Begin
{ Clipping in Z ;}
  If Z>=Zorg then
    Begin
    { What happen if point are behind me? }
     Xf:=319;
     Yf:=200;
    End
  else
     Begin
          Xf := XScreen+((XOrg*Z - X*ZOrg) div
(Z-ZOrg));
          Yf := YScreen+((YOrg*Z - Y*ZOrg) div
(Z-ZOrg));
    End;
End;

BEGIN
  MakeCosTable;
END.
```

las fórmulas de proyección se hace cero, lo que producirá un error de división por cero. La solución a esto se denomina "clipping" o recorte en Z, y consiste en no calcular la proyección



El campo de estrellas.

de aquellos puntos u objetos que se encuentren detrás del observador. Es evidente que dichos puntos no serán visibles.

LA IMPLEMENTACIÓN

Una buena implementación será imprescindible para conseguir un mejor rendimiento del código.

La implementación, puede ser sencilla, o rápida. En el código que acompaña a este articulo se incluyen dos librerías. La primera, es meramente ilustrativa, pero muy transparente, únicamente traduce las fórmulas explicadas anteriormente, de forma que nuestro ordenador las comprenda. En ella se observa la facilidad de implementación de las fórmulas tridimensionales a un punto.

En el listado 1, podemos apreciar la increíble cantidad de cálculos, en coma flotante, que va a realizar el procesador,

BUSCAMOS A LOS MEJORES

- Programadores en C/C++
- Grafistas, diseñadores, dibujantes
- Expertos en lenguaje ensamblador
- Programadores 3D
- Expertos en Sonido
- Músicos con nociones de MOD, MIDI
- Programadores de juegos
- Animadores gráficos
- Infografistas con experiencia en 3D Studio, Photoshop, Deluxe Paint Animation
- Programadores con dominio de lenguajes multimedia: Authorware, Toolbook, Visual Basic, Macromedia Director, etc
- Expertos en comunicaciones

PARA DESARROLLAR

SOFTWARE MULTIMEDIA:

Presentaciones, libros interactivos, programas educativos, sistemas de comunicaciones, centros servidores de datos, videojuegos y mucho más...

SI ERES programador, músico o infografista y tienes ideas o proyectos en estudio de desarrollo ponte ya en contacto con nosotros.

TE OFRECEMOS las mejores condiciones y apoyo para producir tus programas y venderlos en el mercado nacional y extranjero. Formación en nuevas tecnologías y la mejor biblioteca de rutinas gráficas y sonido para desarrolladores.

Aportamos gráficos, rutinas o música, según las necesidades, para complementar cada proyecto.

Si estás interesado en unirte a una de las empresas más punteras en alta tecnología y desarrollo de software, no dejes pasar esta oportunidad. Envíanos carta con tus datos personales (curriculum vitae, con una muestra de tus anteriores trabajos) y un teléfono de contacto a:

DDM DIGITAL DREAMS MULTIMEDIA

DIGITAL DREAMS MULTIMEDIA
Ref. Programadores
C/ Vicente Muzas 15, 1º D - 28043 MADRID
Tf.: (91) 519.23.53 Fax (91) 413.55.77
BBS: (91) 519.75.75 Internet: ddm@servicom.es

para calcular las rotaciones de cada punto tridimensional, pero no olvidemos que se trata de una versión de aprendizaje.

En sucesivos artículos se ira optimizando dicha librería hasta conseguir que sea, lo suficientemente rápida, como para poder manejar, un numero respetable de puntos en tres dimensiones.

No obstante, el lector, no tendrá que esperar tanto, para ver altamente incrementado el rendimiento de dicha librería, si bien, no es el óptimo, ni mucho menos. Como se ha visto a lo largo de este curso, el uso de tablas para calcular, senos y cosenos, agiliza enormemente el proceso de cálculo de las fórmulas, esto es debido, a que el procesador, usa una complicada cantidad de cálculos en coma flotante, para realizar dichas operaciones. Esta será, por supuesto, la primera implementación que se realizará en la librería. Pero además, como cada objeto estará definido por gran cantidad de puntos, sólo será necesario calcular el seno y el coseno de los ángulos de rotación (caída, deriva y balanza) una única vez para todos los puntos que forman un objeto. Y esta será la segunda implementación que vamos a realizar en nuestra librería, como se puede observar en el listado 2.

Una buena implementación será imprescindible para conseguir un mejor rendimiento del código

LAS ESTRELLAS

Sólo se aplica la proyección de los puntos, para calcular el campo de estrellas.

El código que acompaña a este artículo tiene dos ejemplos, uno de ellos, es el campo de estrellas, como se ha comentado al principio del artículo.

Un campo de estrellas, normalmente, no girará sobre ninguno de los tres ejes, por lo tanto, no será necesario aplicar las matrices de rotación a los puntos, con el consiguiente ahorro de cálculos.

El proceso a seguir para realizar un campo de estrellas es el siguiente:

En primer lugar, se calcula de forma aleatoria la posición de cada una de las estrellas en el espacio, posición que será almacenada en una tabla.

En segundo lugar, se creará un degradado para conseguir un bonito efecto 3D. Las estrellas más lejos de la posición del observador, brillarán menos que aquellas más próximas a él.

A continuación, comienza el bucle principal de movimiento del campo. Dicho bucle realizará los siguientes pasos, con cada una de las estrellas:

a- Calcular la proyección de la estrella.
b- Borrar la posición de dicha estrella en el frame anterior.
c- Comprobar que la estrella se encuentra dentro de las dimensiones físicas de la pantalla (en el ejemplo 320x200). Si lo está se dibuja del color de su coordenada z. En caso contrario, se crea una nueva estrella en el campo.
d- Incrementar la coordenada z de la estrella, acercándola, de esta forma, al observador.
e- Comprobar si la estrella está a la altura del observador, en cuyo caso creamos una nueva estrella aleatoria (recordemos que esto es el "clipping" o recorte en z)
f- Almacenar la posición en la pantalla de la estrella, para su posterior borrado (ver paso b).

Y esto es todo lo que hay que hacer, para crear un campo de estrellas. Por supuesto, si se pulsa la tecla ESC, el programa salé del bucle principal y finaliza. Sencillo, ¿no?.

La pantalla de un ordenador tiene 2 dimensiones. Nosotros le daremos la tercera

Los giros sobre cada eje, producen tres matrices de transformación, una, para cada eje

UN OBJETO EN TRES DIMENSIONES

Un ejemplo práctico del uso de la librería.

El otro ejemplo que acompaña a este artículo, ilustra la forma de usar la librería de tres dimensiones, moviendo una esfera definida por 130 puntos. El proceso a seguir, es similar al anteriormente expuesto, con la salvedad, de que, en este caso, sí se aplicarán las fórmulas de rotación sobre los tres ejes. Además, en este ejemplo se emplea un buffer, mientras que en el campo de estrellas, se trabajaba directamente sobre la memoria de vídeo. Los pasos del bucle principal, quedan, entonces, como sigue:

a- Limpiar el buffer.
b- Calcular la rotación de los 130 puntos que conforman la esfera.
c- Calcular la proyección de cada punto.
d- Calcular el color del punto en cuestión, dependiendo de su coordenada z, y ponerlo en el buffer.
e- Esperar un retrazo vertical y volcar el buffer a la memoria de vídeo.

Como se puede observar, es sencillo mover un objeto en tres dimensiones, sobre la pantalla de nuestro ordenador. Queda para el lector, experimentar con el movimiento de la esfera, bien sea modificando las rotaciones, o modificando el centro proyección, o modificando la posición del observador, en fin, posibilidades hay muchas.

FICHEROS EN MVS (II)

José María Peco

Continuando con el tema iniciado el pasado mes, hoy le toca el turno a los métodos de almacenamiento y recuperación de la información en ficheros con organización directa y particionada. Este artículo se complementa con una utilidad que permite intercambiar la información entre los distintos tipos de organización que se estudian

En la entrega anterior fueron enumerados los métodos de almacenamiento y acceso de información bajo el S.O. MVS de IBM. Se vio la organización Secuencial (SAM) y la Secuencial-Indexada (ISAM). El presente artículo estudia el resto de métodos usados para el almacenamiento de la información. La organización directa, método muy usado por los SGDB (Sistemas Gestores de Bases de Datos; o *DBMS Data Base Management System*); la organización particionada, posiblemente la más usada en las grandes instalaciones; y los Ficheros *VSAM* muy usados en todas las instalaciones antes de que se implantara algún gestor como DB2 y/o ADABAS, y que siguen manteniendo totalmente su vigencia.

ORGANIZACIÓN DIRECTA

Este tipo de organización obtiene la dirección física del registro en el dispositivo a partir de la información contenida en el propio registro, para lo cual basta aplicar un algoritmo matemático a la clave para obtener dicha dirección.

Las direcciones obtenidas pueden ser de tres tipos:

- Direcciones relativas de registros (DRR).
- Direcciones relativas de pistas (DRP).
- Direcciones absolutas.

Es decir, que suponiendo que el campo clave de un fichero fuera el DNI, aplicando un algoritmo al valor concreto de cada DNI del fichero, se obtendría la dirección física en la que almacenar cada registro dentro del espacio físico asignado al fichero.

Este tipo de organización se puede definir por la existencia de una correspondencia entre la clave de cada registro y su dirección en el dispositivo, por eso es usado habitualmente por los SGBD para el almacenamiento de los datos.

Función de direccionamiento: Se denomina así a la función que establece la correspondencia entre las claves y sus direcciones en el dispositivo.

DRR : DIRECCIONES RELATIVAS DE REGISTRO

En este tipo de direccionamiento, se considera el espacio asignado al fichero dividido en registros, estando estos numerados consecutivamente, tal y como muestra el ejemplo de la figura 1. Así, si el archivo tiene n registros, y en cada pista pueden caber 5 registros, la dirección relativa de registro dentro de la pista irá desde la 0 hasta la 4.

DRP : DIRECCIONES RELATIVAS DE PISTA

Para este direccionamiento, sólo se consideran numeradas consecutivamente las pistas del espacio asignado al fichero, por lo que la función de direccionamiento proporciona sólo el número de pista en el que ubicar el registro. Existen dos posibilidades para almacenar los datos cuando se usa este tipo de direccionamiento:

1.- Escribir el registro en la primera posición libre de la pista. En este caso, para poder recuperarlo hay que proporcionar la clave además de la DRP.

2.- Proporcionar además de la DRP el número de registro dentro de la pista. En este caso, conocidos ambos datos, se puede obtener directamente la DRR, bastando para ello con una multiplicación y una suma, tal y como muestra el siguiente ejemplo: Se desea conocer la DRR de un registro sabiendo

que la DRP es 2 y el número de registro es el 4 y que se pueden almacenar 5 registros por pista. El resultado seria: 2*5=10 y 10+4=14 por lo que la DRR pedida es 14, pudiéndose comprobar este resultado en la tabla de la figura 1.

Pista \ Reg	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14
3	15	16	17	18	19
...					

Figura 1: Determinar la DRR

DIRECCIONES ABSOLUTAS:

Este direccionamiento necesita un campo de 8 octetos en binario puro para poder recoger la siguiente información:

1 octeto para el número de volumen
2 octetos a ceros binarios
2 octetos para el número de cilindro
2 octetos para el número de cabeza
1 octeto para el número de registro.

La dirección absoluta se obtiene a partir de la DRR por divisiones sucesivas.

Ejemplo: La figura 2 muestra el cálculo de la DRP y la dirección absoluta en un disco 3330 de IBM suponiendo que se haya definido 5 registros por pista y sabiendo que la DRR del registro a calcular es 12345

CLAVES

Las claves en este direccionamiento deberán ser siempre numéricas, y, en el caso de que una clave fuera alfanumérica, será necesario aplicarla una transformación, siendo el método más usual el de las tablas de conversión.

Rango de claves: Es el máximo número de claves que se pueden tener. Se obtiene por diferencia entre la clave más alta y la más baja, siendo un caso habitual que parte de las claves que están incluidas en el rango no se utilicen. Esto quiere decir que el rango es mucho mayor que los números de claves utilizados.

Otro concepto importante es la distribución de las claves que se van a utilizar. Esta distribución es uniforme si la proporción de claves utilizadas es igual en cualquier parte del rango; en caso contrario se dice que no es uniforme, siendo ésta la más normal.

SINÓNIMOS

Son registros de claves diferentes pero que al aplicarles la función de direccionamiento resulta la misma dirección. Este problema no surge únicamente con el método de división-resto visto anteriormente, sino que surge con cualquier método que se use.

Los sinónimos en sí mismos pueden no ser un problema, dependiendo del tipo de direcciones que se generen. Así, si se esta generando DRR, todos los registros que tengan la misma dirección serán sinónimos, y a la vez excedentes, porque todos los registros sinónimos no se pueden grabar en la misma dirección.

Pero, si se estuviera generando DRP, todos los registros que pertenezcan a la misma pista serán sinónimos, pero no habrá excedentes hasta que se llene la pista.

Una función de direccionamiento será peor o mejor, dependiendo de que genere mayor o menor número de excedentes. En el caso de que se generen muchos excedentes, lo que se suele hacer es ampliar el número de direcciones de un fichero, dando más espacio del que realmente se necesita. Llevando al limite esta posibilidad se podría reservar espacio para todas las direcciones del rango, lo cual conduciría a una infrautilización del fichero.

Factor de empaquetamiento: Una medida de esta técnica es lo que se denomina factor de empaquetamiento, definiéndose como el cociente que resulta al dividir el número de registros utilizados por el número de registros reservados. Lo normal es usar factores de empaquetamiento entre el 80 y 85 %.

Formato de los registros: La organización directa admite todos los tipos de formatos, sin embargo, los registros son considerados siempre como desbloqueados. Por tanto, si se desea bloquear registros, será el propio informático el que se plantee y resuelva el problema, generando bloques y asignándoles una clave a la que aplicar el algoritmo de direccionamiento. Esto último es lo que realizan los distintos gestores de bases de datos que utilizan este tipo de organización.

En este sentido, cabe recordar el hecho comentado al tratar el tema de DB2, en el que el espacio físico asignado al fichero (tabla) se encontraba dividido en paginas, pudiendo contener cada una de estas páginas varios registros lógicos. Pues bien, el hecho que se desea resaltar es que si bien el registro lógico, desde el punto de vista del usuario , es el registro de *clientes*, por ejemplo, desde el punto de vista del SGDB, el registro lógico esta constituido por una página, no pudiendo aplicarse el concepto de bloqueo visto en

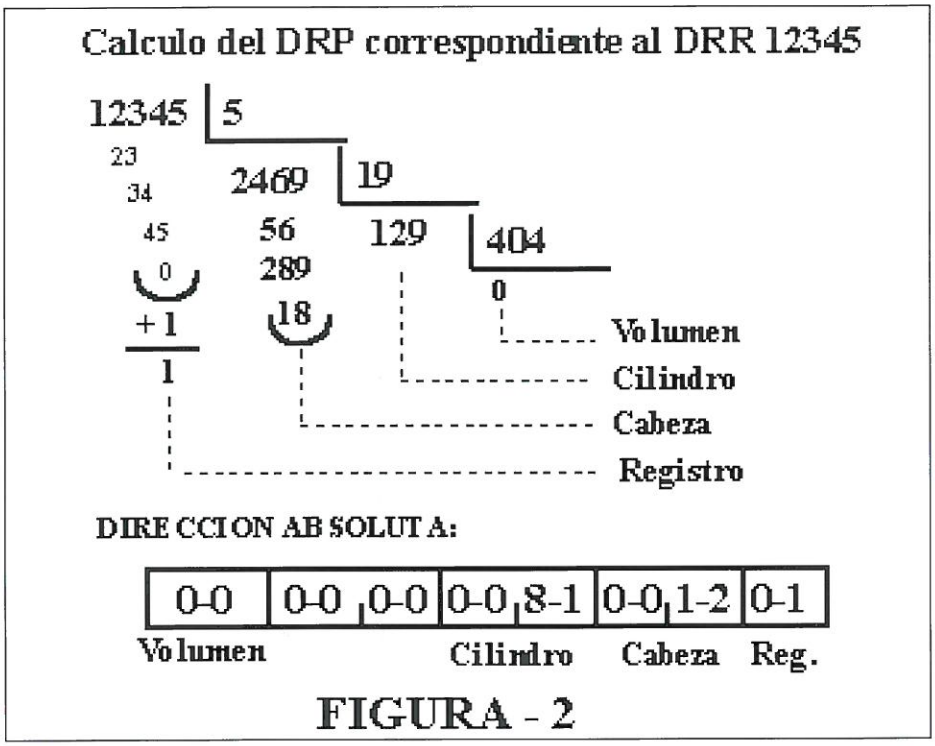


FIGURA - 2

el artículo anterior, por ser cada página un único registro lógico para el gestor.

PROCESOS

Carga: Es siempre al azar. Se produce progresivamente en el tiempo.

Adiciones: Son siempre al azar, e iguales al proceso de la carga. Al aumentar el número de registros se incrementa el *factor de bloqueo*

Recuperación: También se realiza mediante proceso al azar. Al aumentar el número de excedentes disminuye la eficacia.

ORGANIZACIÓN PARTICIONADA

Este tipo de organización nace con el sistema operativo DOS/VS, siendo junto a la secuencial las organizaciones más usadas en una gran instalación de MVS en las labores cotidianas.

El nombre de esta organización se debe a que tiene la información dividida en dos partes o áreas:

- Área de directorio
- Área de miembros.

El Área de directorio hace las veces de un diccionario en el que se encuentran referenciados cada uno de los miembros del fichero, pues contiene los nombres y direcciones de cada uno de ellos; y, el Área de miembros recoge la información de interés para el usuario.

Un miembro es el equivalente de un fichero secuencial, con la salvedad de que el conjunto formado por todos estos ficheros secuenciales configuran un único fichero. Pero, como las características se definen a nivel de fichero, todos los miembros tendrán la misma longitud de registro, formato, etc.

Se podría equiparar esta organización a la de una biblioteca en la que el área de directorio representa el índice, y el área de miembros estaría representando a cada uno de los libros físicos de la biblioteca, eso si, todos los libros deberían tener el mismo formato o longitud de registro, aunque como es lógico, todos tendrían distinto contenido, autor, número de páginas, etc.

DIRECTORIO

Es un conjunto de registros lógicos agrupados en bloques de 256 bytes, lo cual quiere decir que en un bloque deberán grabarse el máximo número de registros lógicos enteros que quepan.

En todo bloque, el primer octeto contiene el número de octetos ocupados del bloque.

Los registros lógicos pueden contener información opcional, por lo que su longitud varia desde 12 octetos como mínimo, hasta 74 octetos como máximo. En los primeros doce octetos viene el nombre del miembro, y la dirección del mismo en el área de miembros.

Todos los registros en el área de directorio se encuentran clasificados alfabéticamente, tal y como muestra la figura 3.

MIEMBROS

Los miembros son ficheros que no tienen relación alguna con el resto de miembros del fichero, es decir, son independientes entre sí. Se encuentran organizados consecutivamente, lo cual quiere decir que la organización es compacta y que, por tanto, no hay huecos libres entre los miembros.

Admite cualquier tipo de formato, de pistas y de registros.

Para recuperar un miembro, se debe buscar el nombre del mismo en el área de directorio del fichero; y una vez obtenida su dirección, recuperar secuencialmente la información contenida en el mismo.

Para añadir miembros, el propio sistema genera áreas adicionales y reorganiza el directorio para colocar alfabéticamente el nombre y la dirección de los nuevos miembros en el lugar que les corresponda.

Los miembros que se dan de baja, no desaparecen físicamente del directorio en el momento de darle de baja, sino que el miembro permanece hasta que se reorganice físicamente el área de miembros.

Cuando el área del directorio se llena, no se pueden añadir nuevos miembros por falta de sitio en el directorio, siendo necesario reorganizar toda la librería, tanto de miembros como directorio.

PARTICULARIDADES DE ESTA ORGANIZACIÓN

• Como ya se ha dicho, el fichero esta formado por el conjunto de las dos áreas citadas, por lo que a la hora de asignar recursos para un trabajo (*ALOCAR*), si ya existe el fichero, se debe poner disposición *OLD* o *SHARE*, aunque lo que se desee sea crear un miembro, tal y como se puede ver en el *JCL* que se encuentra en el disquete que acompaña a la revista.

• El concepto de reorganización citado anteriormente consiste en eliminar físi-

IMPLEMENTACIÓN DE UN DEVICE DRIVER (II)

Pere Ridau, Jordi Freixenet, Enrique F. Torres

Se presenta este mes el último de los artículos dedicados a la implementación de drivers sobre Linux. En esta ocasión, se expone el funcionamiento de los drivers para dispositivos de bloques, presentando como ejemplo un *ramdisk*

Entre la diversidad de periféricos existentes hoy en día, adquieren especial relevancia para el sistema operativo (S.O.), los dispositivos de bloques. Un disco flexible, un disco duro, una cinta magnética o un *ramdisk*, son ejemplos típicos. Este mes se analizan los drivers que gestionan estos dispositivos.

DISPOSITIVOS DE BLOQUES

Se denominan dispositivos de bloques a aquellos para los cuales la unidad de transferencia de información entre el S.O. y el dispositivo es el bloque. Estos tienen la posibilidad de contener un sistema de ficheros y por tanto, pueden ser montados dentro del árbol de directorios.

OPERACIÓN CON DISPOSITIVOS DE BLOQUES

Existen dos formas básicas de acceso a un dispositivo de bloques:

- *Directo:* Cuando un proceso abre el fichero de dispositivo asociado y opera directamente con él como si se tratase de un fichero ordinario. En este caso, el dispositivo se comporta como un fichero de tamaño fijo e igual al producto del número de bloques que contiene por el tamaño de bloque. La Figura 2 muestra un ejemplo de este tipo de acceso.
- *Indirecto:* Cuando un proceso opera con un fichero contenido en el árbol de directorios de un dispositivo de bloques montado previamente. Éste es el uso más común (ver Figura 3).

En ambos casos, para acceder al dispositivo, Linux aplica una técnica especial de buffering denominada *buffer cache.*

LA BUFFER CACHE

En el acceso a un dispositivo de bloques, el S.O. siempre resuelve las peticiones generadas bloque a bloque. Dada la frecuencia de estas operaciones (sobre todo en el acceso al sistema de ficheros), interesa minimizar el tiempo efectivo de lectura/escritura de un bloque de datos. Con el fin de reducir este tiempo, Linux intenta mantener los bloques más utilizados en un conjunto de buffers ubicados en memoria física, denominado *buffer cache.*

Dada la limitación de espacio, no es posible contener en memoria todos los bloques de todos los dispositivos, por lo que Linux sólo mantiene un subconjunto de ellos. Cuando el S.O. necesita un bloque (b) de un dispositivo (d) (para poder resolver, por ejemplo, una llamada al sistema *read*) ejecuta la primitiva *buffer=bread(d,b)*. Esta primitiva utiliza la rutina *getblk(d,b)* para comprobar si el bloque solicitado se halla contenido en alguno de los buffers, retornándolo en caso afirmativo. Si el bloque no ha sido hallado, *getblk* devuelve un buffer libre y *bread* ejecuta una llamada al driver del dispositivo para que lea el bloque, retornando finalmente el buffer actualizado. Dado que el número de buffers de la cache es finito, puede darse el caso de no encontrarse ningún buffer libre (es decir que todos los buffers estén "mapeando" otros bloques). En este caso, el S.O. escogerá el buffer que haga más tiempo que no se utiliza (algoritmo *LRU*), actualizará el bloque asociado en caso de que el contenido del buffer difiera del contenido del dispositivo, y reciclará dicho buffer.

De forma paralela, cuando el S.O. necesita escribir un bloque, ejecuta la

primitiva *mark_buffer_dirty*. Ésta sitúa el bloque en la cache y lo marca como no actualizado sobre el dispositivo, de forma que si posteriormente es necesario reciclar el buffer, el bloque sea escrito físicamente sobre el dispositivo.

Con el funcionamiento descrito, en el mejor de los casos, los bloques solicitados se encontrarán en la cache, y las peticiones se resolverán a velocidad de memoria. En el peor de los casos, el bloque deberá ser leído por el driver sobre un buffer, y la petición se resolverá a velocidad del dispositivo (ver Figura 1).

El lector interesado en profundizar en el estudio de la buffer cache, puede leer el capítulo 3 de *[Bach86]*.

OPERACIONES UNIFORMES

Tal y como se comentó en el anterior artículo, los procesos acceden a los dispositivos de bloques a través de un conjunto de operaciones uniformes (llamadas al sistema *open*, *close*, *read*, *write*, *lseek*, *ioctl/fcntl*). Una operación *read* por ejemplo, generará en última instancia una secuencia de peticiones de lectura de bloques. Si los bloques solicitados son hallados en la cache, la operación se resolverá instantáneamente. Si por el contrario, alguno de estos bloques no es hallado, deberá ser leído por el driver y situado en un buffer de la cache. Así pues, el driver debe proveer una rutina que realice la operación física de lectura de un bloque sobre un buffer asignado por la cache. De forma paralela, cuando el contenido de un buffer de la cache deba ser escrito en el dispositivo (por ejemplo, porque el buffer se recicle y contenga un bloque que deba ser actualizado), dicha rutina deberá encargarse de volcar el contenido del buffer sobre el dispositivo. Se puede observar pues que la función del driver es la de proveer una rutina (denominada *strategy*) que se encargue de leer bloques físicos sobre buffers de la cache, o de escribir el contenido de éstos sobre los bloques asociados dentro del dispositivo. La pregunta que surge en este punto es: "¿cómo sabe el sistema operativo qué driver debe llamar para llenar/vaciar un buffer?" Para responder esta cuestión deben diferenciarse dos casos:

- acceso directo al dispositivo
- acceso indirecto a través del sistema de ficheros que contiene.

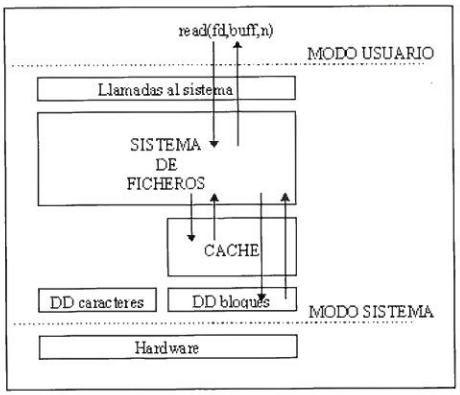


Figura 1.

```
# ls -alis > /dev/fd0
#cat /dev/fd0
```

Figura 2.

ACCESO DIRECTO. MAPEO DE LAS OPERACIONES UNIFORMES SOBRE LAS OPERACIONES ESPECÍFICAS

Al igual que en los drivers de caracteres, la estructura *file_operations* contiene, para cada operación uniforme, la dirección de la operación específica asociada. No obstante, imagínese dos drivers de dos dispositivos de bloques: un *floppy* y un *ramdisk*. Las operaciones específicas asociadas a la operación uniforme *read* tendrían, respectivamente, el siguiente aspecto:

- *read_blk_fd(inode, file, count)*
- *read_blk_rd(inode, file, count)*

Ambas rutinas se comportan de manera idéntica. Dado el puntero de L/E de un fichero (contenido en la estructura *file*) y un número de bytes a leer (*count*), se calculan la secuencia de bloques que deben ser leídos para satisfacer la operación y los piden a la cache. Las dos rutinas son idénticas, la única diferencia radica en el valor del identificador de dispositivo (d) referido en la llamada a *getblk(d,b)* que en el caso de la primera contiene el par (*major,minor*) del *floppy*, y en la segunda el del *ramdisk*.

Parece lógico pues que, en lugar de obligar al programador a proporcionar las rutinas específicas *read_blk_dev* y *write_blk_dev* para cada dispositivo de bloques, el *kernel* proporcione dos rutinas genéricas: *block_read* y *block_write* que realicen dicha función. Debido a esto, el campo *read* de la estructura *file_operations* contendrá la dirección de la rutina *block_read*, y el campo *write*, la dirección de la rutina *block_write*.

La Figura 4 muestra la secuencia de llamadas desencadenada por una operación *read*, desde que es ejecutada por un proceso, hasta que se accede a la rutina *strategy* del driver. A continuación se detalla la funcionalidad de cada una de las rutinas:

- *_syscall*

Rutina *trap dispatcher*. Toda llamada al sistema es interceptada por esta rutina, que la identifica y ejecuta la rutina interna que implementa dicha llamada al sistema.

- *sys_read(int fd, char *buff, int count)*

Rutina interna que implementa la operación uniforme o llamada al sistema *read*. Esta rutina tiene los mismos parámetros que la rutina de librería asociada a la llamada al sistema y se encarga de ejecutar la rutina específica.

- *block_read(struct inode *inode, struct file * filp, int count)*

Rutina genérica ejecutada por la rutina *sys_read* (rutina independiente de dispositivo) y proveída por el *kernel*. Realiza la función de rutina específica de dispositivo (*read_blk_dev*).

Identifica los bloques que deben ser leídos para satisfacer una operación *read* y los pide a la cache. Seguidamente ejecuta la rutina *ll_rw_block* para leer aquellos bloques que no fueron hallados en la cache.

- *getblk(dev, block, size)*

Retorna un buffer que puede estar en dos estados:

Válido: en este caso el buffer contiene el bloque (*block*) del dispositivo (*dev*).

Inválido: el bloque solicitado no se

```
# mount /dev/fd0 /mnt
# ls /mnt
        f1
# progr
```

```
/* progr c*/
main()
{
int fd;
char buf[10];

fd=open("/mnt/f1",O_RDONLY);
read(fd,buf,10);
close(fd);
}
```

Figura 3.

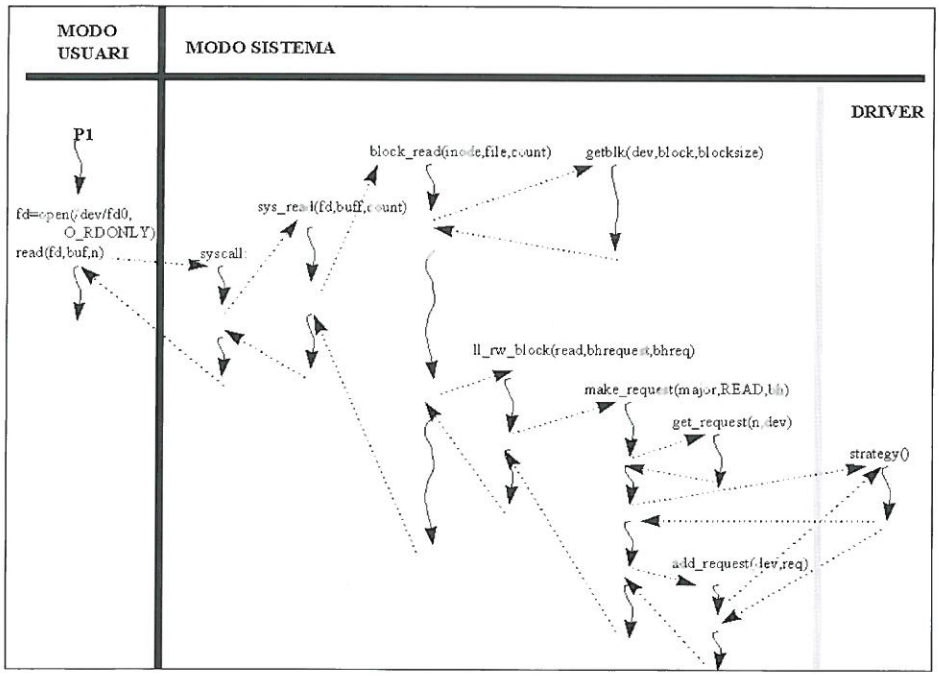


Figura 4.

halló en la cache. El buffer retornado es un buffer libre sobre el cual se podrá leer del dispositivo el bloque deseado.

- *ll_rw_block(int rw, int nr, struct buffer_head *bh[])*

Bh es un array de buffers que contiene *nr* buffers que deben ser leídos o escritos según la operación indicada en *rw* y que no estaban en la cache.

- *make_request(int major, int rw, struct buffer *bh)*

Obtiene una petición vacía ejecutando *get_request*, la rellena y la sitúa en la cola asociada al dispositivo (*blk_dev[major].current_request*; ver Figura 5) utilizando la función *add_request*.

- *struct request * get_request(int n, int dev)*

Esta función retorna una estructura petición libre, que será rellenada para construir una petición.

- *add_request(struct blk_dev_struct * dev, struct request *req)*

Si la cola asociada al dispositivo está vacía, encola la petición y ejecuta la rutina *strategy*; en otro caso simplemente encola la petición. En ambos casos la petición se sitúa de forma ordenada según el algoritmo del ascensor (algoritmo de planificación que intenta mejorar el rendimiento del dispositivo).

Se analizan a continuación los pasos que sigue una operación hasta ser resuelta.

Dentro del *inodo* (parámetro de *block_read*) existen dos identificadores de dispositivo:

- *i_dev*: Identifica al dispositivo que contiene el *inodo*.
- *r_dev*: Identifica al dispositivo representado por el *inodo*, en caso de tratarse del *inodo* de un fichero de dispositivo.

Si se opera sobre el fichero de dispositivo de un *ramdisk* y dicho fichero está situado en el disco duro, *i_dev* es el identificador de dispositivo del disco duro y *r_dev* el del *ramdisk*.

Cuando *block_read* solicita bloques al dispositivo (*ll_rw_block*) construye un array de buffers indicando entre otras cosas el identificador de dispositivo (*r_dev*) y el número de bloque. *R_dev* es una codificación del par (*major,minor*), de forma que *ll_rw_block* obtiene el *major* y realiza, para cada buffer, una petición de L/E (*make_request*). Utilizando el *major* para indexar en la tabla *blk_devs* el S.O. halla la cola de peticiones pendientes asociada al dispositivo y la dirección de la rutina *strategy* (almacenada allí durante la inicialización del driver). Finalmente, la petición es encolada por la rutina *add_request* (ejecutada desde *make_request*) que recibe como parámetro el elemento *blk_dev[major]*, de forma que puede acceder a la cola del dispositivo y a la rutina *strategy*.

ACCESO A TRAVÉS DEL SISTEMA DE FICHEROS. MAPEO DE LAS OPERACIONES UNIFORMES SOBRE LAS OPERACIONES ESPECÍFICAS

La Figura 3 muestra el código de un programa (progr.c) que lee 10 bytes de un fichero (f1) contenido en un disco flexible previamente montado.

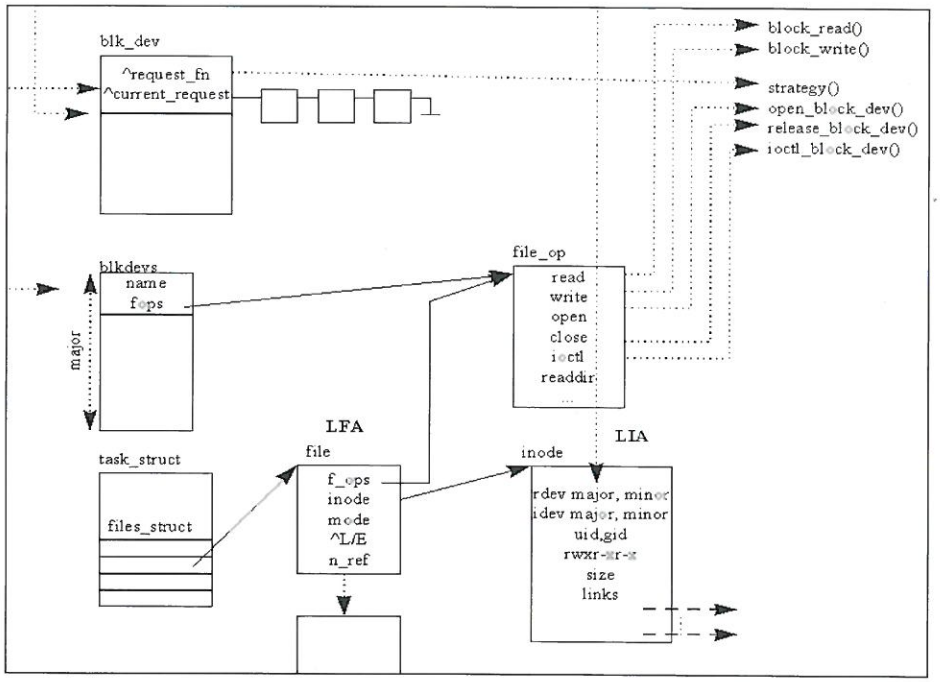


Figura 5.

CORREO LECTORES

Para cualquier duda referente a los temas/artículos tratatos en la revista, escriba una carta a:

Sólo Programadores
Referencia: *Correo Lectores*
TOWER COMMUNICATIONS

CÓMO SUSCRIBIRSE A

SÓLO PROGRAMADORES

Revista práctica para usuarios de PC

Suscríbase enviando este cupón por correo o fax (91) 661.43.86, o llamando al teléfono (91) 661.42.11 Horario 9 a 14 y 15:00 a 18:00 h.

Deseo suscribirme a la revista SÓLO PROGRAMADORES acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año (12 números) **por sólo** 11.950 ptas. (ahorro 20%). ☐ Estudiantes carreras técnicas: 8.950 ptas. (ahorro 40%)

*** ESTA OFERTA ANULA LAS ANTERIORES, DESCUENTOS NO ACUMULABLES.**

Nombre y apellidos.....Domicilio.....

Población.....C.P .....Provincia.....Telf.....Profesión.....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº ____ ____ ____ ____

Fecha de caducidad de la tarjeta.....Nombre del titular, si es distinto.....

☐ Domiciliación bancaria.
Señor Director del banco .....
Población .....
Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta de ahorro número.....
el recibo que le será presentado por TOWER COMMUNICATIONS, S.R.L.
como pago de mi suscripción a la revista SÓLO PROGRAMADORES.

CODIGO CUENTA CLIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Firma:

☐ Contra-reembolso del importe más gastos de envío.
☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.
☐ Giro Postal (adjunto fotocopia del resguardo).

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Aragoneses, 7
28100 Pol. Ind. ALCOBENDAS (Madrid)

Después de la ejecución del *open*, las estructuras de datos internas quedan en la situación reflejada en la Figura 7. El descriptor de fichero (*fd*) devuelto por *open*, apunta a un elemento de la *LFA* (Lista de Ficheros Abiertos) que a su vez apunta a la estructura *file_operations*. A diferencia del caso de acceso directo, la estructura *file_operations* apuntada no es la del driver, sino la asociada dentro del *VFS* al sistema de ficheros contenido en el disco montado (*VFS: Virtual File system Switch*: característica que permite a Linux operar simultáneamente con diferentes sistemas de ficheros como minix, msdos, ext, ext2, etc). Así por ejemplo, si el disco montado es un disco de MS-DOS, el campo *read* de la estructura *file_operations*, contendrá la dirección de la rutina *msdos_file_read*. Ésta implementa la rutina dependiente del sistema de ficheros asociada a la operación uniforme *read*.

Cuando un proceso realiza una llamada al sistema *read* (Figura 6), se ejecuta la rutina *msdos_file_read*, que pedirá la información necesaria a la cache. Si la información no es hallada en la cache, se desencadenará una secuencia de llamadas a diferentes rutinas hasta llegar a la ejecución de la rutina *strategy* del driver asociado que rellenará los buffers necesarios con los bloques solicitados.

¿Cómo sabe el *kernel* qué rutina *strategy* debe ejecutar? De forma similar al caso anterior, la rutina *msdos_file_read* recibe como parámetro el *inodo* y con él los campos *i_r_dev*. En el momento de solicitar los bloques a la cache se utilizará como identificador de dispositivo *i_dev*, ya que éste identifica al dispositivo que contiene el *inodo* y, por tanto, el que contiene el fichero. El resto del procedimiento es similar al caso de acceso directo.

DRIVERS DE BLOQUES

El esquema de los drivers de bloques es similar al de los drivers de caracteres analizado en el anterior artículo. El driver debe proveer, las rutinas específicas asociadas a las operaciones uniformes. Tal y como se introdujo anteriormente, las rutinas especificas *read_block_dev* y *write_block_dev*, ya vienen implementadas dentro del *kernel* en forma de

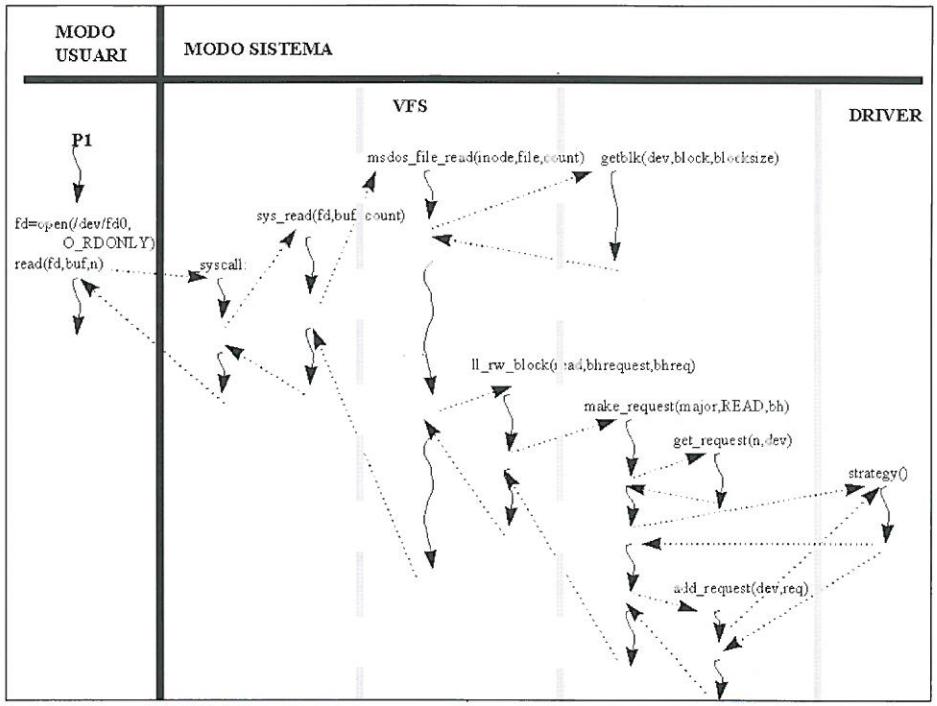


Figura 6.

dos rutinas genéricas: *block_read* y *block_write*. Un driver de bloques debe proveer además, una rutina *strategy* responsable de vaciar o rellenar los buffers de la cache. Se comentará a continuación aquellas rutinas relevantes que presenten alguna diferencia respecto a los drivers de caracteres:

- *int register_blkdev(unsigned int major, const char * name, struct file_operations *fops)*

Punto de entrada: Durante la inicialización del driver. Debe ser llamada desde la rutina *init_blk_dev*.

Función: Registra el driver dentro de las estructuras de datos del S.O. Actualiza la variable *blkdevs[major]* (ver Figura 7) para que contenga el nombre del dispositivo identificado por *major* y la estructura *file_operations* asociada. Esto permitirá que durante la apertura de un fichero de dispositivo, y una vez carga-

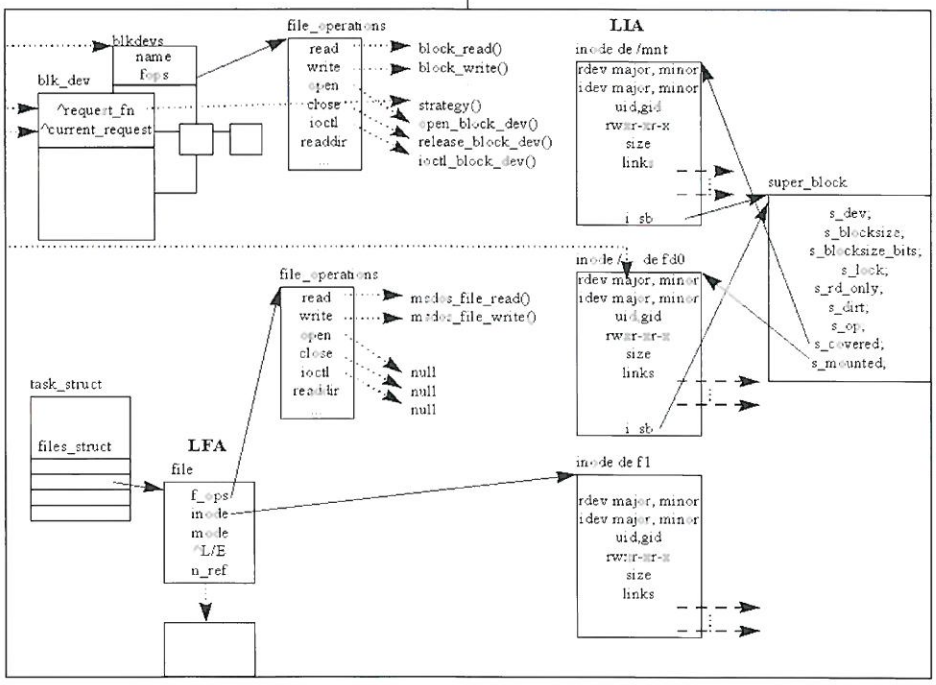


Figura 7.

do el *inodo* en la *LIA* (Lista de Inodos Activos), el S.O. obtenga el *major* (a partir del campo *r_dev* del *inodo*) e indexando sobre el array *blkdevs*, localice la estructura *file_operations*. Una vez localizada, podrá actualizar el enlace necesario (*f_ops*) entre el elemento de la *LFA* asignado y la estructura *file_operations*.

Parámetros:

major: Identificador de dispositivo.

name: Nombre del dispositivo.

fops: Dirección de la estructura *file operations*.

Retorno:

0: La operación ha acabado satisfactoriamente.

<0: La operación ha finalizado erróneamente.

- *init_blk_dev(long memstart, long memend)*

Punto de entrada: Durante el proceso de *boot*. Es necesario añadir una llamada a esta rutina en el fichero */usr/src/linux/drivers/block/ll_rw_blk.c*, dentro de la rutina *blk_dev_init*.

Función: Inicialización del driver. Es responsable de inicializar las variables internas del driver, y reservarle memoria dinámica en caso de ser necesario.

Debe realizar las siguientes operaciones:

1. Registrar el driver ejecutando *register_blkdev*.
2. Actualizar *bl_dev[major].request_fn* para que apunte a la rutina de *strategy*.
3. Construir un array de dos elementos, hacer que ambos contengan el tamaño de un bloque en bytes y actualizar *blk_size_size[major]* para que contenga la dirección de dicho array.

Parámetros: En la actualidad el programador del driver es libre de escoger qué parámetros debe pasar a la rutina, ya que él es el responsable de programar la llamada a esta rutina. No obstante lo más usual es trabajar con los siguientes parámetros:

memstart: Última dirección de memoria utilizada. A partir de ésta, el driver puede reservarse el espacio que necesite y retornar la nueva dirección de final de memoria utilizada.

memend: Contiene la última dirección de memoria disponible.

Retorno: A libre elección del programador. Usualmente, la última dirección de memoria utilizada (el valor de *memstart* incrementado según el tamaño de la memoria reservada por el driver).

- *strategy()*

Punto de entrada: Se ejecuta cada vez que hay que cargar información de un dispositivo sobre un buffer de la cache, o cada vez que hay que volcar el contenido de un buffer sobre el dispositivo asociado.

Función: Es responsable de dar servicio a las peticiones situadas en la cola *blk_dev[major].current_request*. En primer lugar debe ejecutar la macro *INIT_REQUEST* (definida en */usr/src/linux/drivers/ block/blk.h*) para realizar algunos chequeos de error. Si la gestión de la E/S no se realiza a través de interrupciones, la rutina procesará todas las peticiones y retornará inmediatamente. Si por el contrario, la gestión de E/S es por interrupción, programará la primera petición y acto seguido retornará. En este caso, la rutina de interrupción será la encargada de iniciar la próxima petición cuando la actual finalice. En ambos casos, se ejecutará *end_request(x)* al finalizar una operación, con *x=1* si todo ha ido correctamente y *x=0* en cualquier otro caso.

Para poder procesar las peticiones, es necesario conocer su estructura:

cmd: Indica la operación a realizar (*READ / WRITE*).

sector: Primer sector a operar.

current_nr_sectors: Número de sectores consecutivos a sector que deben ser operados.

buffer: Dirección del buffer donde deben leerse o escribirse los datos.

Parámetros: -

Retorno: -

- *open_blk_dev(struct inode *inode, struct file *file)*

Su función es idéntica a la de la función equivalente en los drivers de caracteres. La única diferencia radica en que ésta también se ejecuta cuando se monta un dispositivo de bloques.

- *release_blk_dev(struct inode *inode, struct file * file)*

De forma paralela al caso anterior, esta rutina se comporta como su equivalente en los drivers de caracteres, con la diferencia de que también es ejecutada al desmontar un dispositivo de bloques.

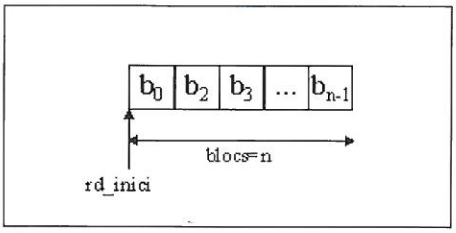


Figura 8.

RAMDISK: UN EJEMPLO

Un disco puede definirse en general como un dispositivo de almacenamiento de información en unidades de bloques. Los bloques están formados por uno o más sectores y vienen numerados linealmente (bloque 0, bloque 1, ..., bloque n). A bajo nivel, existen dos operaciones básicas sobre un disco:

- lectura de un bloque
- escritura de un bloque

Si bien un floppy utiliza un medio de almacenamiento magnético y permanente, en el caso del *ramdisk*, se utiliza como medio de almacenamiento la memoria RAM. Así pues, puede considerarse un *ramdisk* como un array de bloques ubicados en memoria (ver Figura 8).

Dada la simplicidad del dispositivo, para construir el driver sólo es necesario programar dos funciones:

- *rmd_init(long mem_start, unsigned char sectpblk, int blocks)*

Parámetros:

mem_start primera posición libre de memoria.

sectpblk número de sectores por bloque.

blocks tamaño del disco en bloques.

Función: Inicialización del driver. Registra el driver, almacena la dirección de la rutina *strategy* e inicializa las siguientes variables:

rd_sector: tamaño del disco en sectores.

rd_inicio: dirección de memoria donde comienza el *ramdisk*.

Posteriormente, inicializa el espacio de memoria ocupado por el *ramdisk* a '\0' y crea un array con dos posiciones que contienen el tamaño (en bytes) de un bloque. La dirección del anterior array se almacena sobre *blksize_size[MAJOR_NR]* (estructura de datos interna del *kernel*).

El espacio utilizado por el disco se sitúa a partir de *mem_start*, y ocupa *sectpblk*blocks*512* bytes (un sector

```
#define RD_MAJOR 31              /* Escogemos como major del dispositivo el 31 */

char *rmd_inicio;                /* 1ª dirección de memoria del ram disk */
unsigned long rd_sector;         /* Tamaño del ram disk en sectores */
static int rmd_blksize[2];

long rmd_init(long mem_start, unsigned char sectpblk, unsigned int blocks)
    {
    long i;

    if (register_blkdev(RD_MAJOR,"rmd", file_operations)
        {
        Error major duplicado;
        Salir;
        }
    blk_dev[RD_MAJOR].request_fn=rmd_strategy;
    rmd_sector=blocks*sectpblk;
    rmd_inicio=mem_start;

    Asignar el valor '\0' a cada byte del ram disk;
    rmd_blksize[0]=rmd_blksize[1]=sectpblk*512;
    blksize_size=[RD_MAJOR].rmd_blksize;

    return(rd_inicio+rd_sector*512);
    }

void rmd_strategy(void)
    {
    for (;;)
        {
        iniciar_petició;
        if ((dispositivo correcto) && (sectores_a_leer_existen))
            {
            switch(operación_de_la_peticion)
                {
                case READ:      Copiar_los_sectores_del_ramdisk_al_búffer;
                                petición_correcta;
                                break;
                case WRITE:     Copiar_el_contenido_del_búffer_a_los_sectores;
                                petición correcta;
                                break;
                default:        petición_erronea;
                }
            }
        else petición_erronea;
        }
}
```

Figura 9.

ocupa 512 bytes). *Rmd_init* retorna la nueva posición en que comienza la memoria libre.

• *rmd_strategy*

Comprueba que el *minor* del dispositivo pertenece al rango de los gestionados por el driver, y que los sectores que forman el bloque a leer/escribir existan. Obtiene una petición de E/S de la cola *blk_dev[MAJOR_NR].current_request* y la procesa. La macro *CURRENT* definida en *blk.h* permite referenciar los campos de la petición actual, de forma que *CURRENT->cmd* indica, por ejemplo, la operación a realizar. La rutina no retorna hasta que todas las peticiones encoladas hayan sido procesadas.

La Figura 9 muestra el pseudocódigo del driver propuesto. El código real se adjunta en el disco de la revista.

NORMAS PARA LA IMPLEMENTACIÓN

Se detallan a continuación una serie de normas a seguir en la implementación de un driver.

BIBLIOGRAFÍA

[Johnson93] Linux kernel Hacker's Guide. Michael K. Johnson. 1993. johnson@sunsite.unc.edu

[Pajari92] Writing Unix Device Drivers. George Pajari. Addison Wesley. 1992.

[Leffler89] The design and Implementation of the Unix Operating System. Samuel J. Leffler, MarsharII kirk mcKusick, Michael J.Karels, Jon S. Quaterman. Addison Wesley. 1989.

[Bach86] The Design of the Unix Operating System. Maurice J. Bach. Prentice Hall International. 1986.

1. Al inicio del fichero fuente del driver añadir las siguientes líneas:

#define MAJOR_NR 31
#include "blk.h"

dónde 31 es el *major* del dispositivo.

2. Modificar el fichero */usr/src/linux/drivers/block/blk.h* añadiendo después de la línea *#ifdef MAJOR_NR* y protegido por un *#elif(MAJOR_NR==31)* las siguientes definiciones:

#define DEVICE_NAME "rmd"
#define DEVICE_REQUEST rmd_strategy

Otras definiciones innecesarias para el *ramdisk*, pero útiles para otros dispositivos son:

#define DEVICE_INTR irq_fn

dirección de la rutina de interrupción asociada al dispositivo.

Para más información consultar *[Johson93]*.

EJEMPLO DE UTILIZACIÓN

Se puede comprobar el funcionamiento del driver mediante el siguiente procedimiento:

1. Se formatea el *ramdisk* eligiendo uno de los sistemas de ficheros soportados por Linux (*ext2* en el ejemplo):

 # mke2fs /dev/rd1

2. Se monta el disco sobre el árbol de directorios:

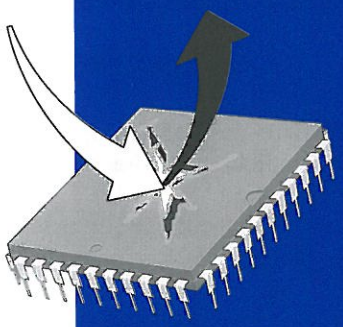
 # mount -text2 /dev/rd1 /mnt

3. Se opera sobre el *ramdisk* utilizando el directorio */mnt*:

$ echo hola > /mnt/f1
$ cat /mnt/f1
hola
$

EL ANÁLISIS SEMÁNTICO

Daniel Navarro



El compilador está prácticamente perfilado. Esto quiere decir que, aunque queden muchas decisiones por tomar, sólo las referentes al análisis semántico afectan al lenguaje, o dicho de otra forma, una vez concluida esta fase, el lenguaje ya estará implementado con todas sus reglas, ya que la fase de síntesis siguiente sólo afectará a la calidad del código generado.

A modo de breve resumen, se recordará a continuación la situación en la que está el compilador detenido.

El análisis sintáctico está completo, lo que quiere decir que, para cualquier programa de entrada, ya se comprueban los dos primeros niveles de reglas; léxicamente, el programa debe tener sólo piezas pertenecientes al lenguaje y sintácticamente, el programa será, a priori, una secuencia válida de dichas piezas. En resumen, está ya aplicada la gramática, lo que asegura que un programa que pase a la siguiente fase lo hará partiendo de la base de que respeta la sintaxis BNF que se había definido.

Ahora bien, en ningún sitio se ha hecho hasta ahora caso al nombre de los datos (recuérdese que en Letra se llamaba datos a las variables), ni las tablas, ni el valor de los enteros, etc. Sólo se examinaba si venia un dato donde debía venir (una pieza *p_identificador* cualquiera, sin importar cual fuera el nombre de dicho dato) o si venía un número entero (una pieza *p_entero*, sin importar tampoco cual fuera su valor).

Todos los objetos definidos en el programa deben ser registrados por el compilador, que los anotará en la tabla de objetos.

LA TABLA DE OBJETOS

En Letra hay tres tipos diferentes de objetos definibles por el usuario: datos, tablas de datos y etiquetas. Todos ellos son objetos que se declararán en alguna parte del programa y pueden ser accedidos desde otras. No deben confundirse con otros elementos del programa como son las constantes (lógicas, números enteros o literales), pues éstas no tienen las características que poseen los objetos, sino un valor fijo.

La tabla de objetos es una estructura de datos interna del compilador construida como una serie de registros con diversos campos, donde los más comunes son:

- Información sobre que TIPO de objeto se registra. Según de qué objeto se trate, el resto de los campos pueden variar, ya que no todos los objetos precisan guardar la misma información; por ejemplo, en el registro de una tabla hace falta tener un campo para especificar la(s) dimensión(es) de ésta, mientras que en el de un dato no tiene sentido.
- El NOMBRE del objeto (el identificador del dato, tabla o variable). Este es el campo más importante, ya que mediante él se podrá saber cuándo se vuelve a hacer referencia a dicho objeto dentro del programa.
- Información sobre el estado del objeto en un punto determinado de la compilación del programa. Por ejemplo, si un lenguaje permite (como es el caso de Letra) que un objeto sea utilizado antes de haberse declarado, el compilador indicará esta situación en un campo donde se especifique si un objeto ha sido DECLARADO (true/false). Si, además, se quiere que el compilador avise cuando un objeto se ha declarado y después no se usa (Lo que se suele advertir mediante un "Error leve" o "Warning") se hará manteniendo en cada registro de objeto un campo denominado USADO (true/false).

Durante la fase semántica, un compilador resuelve todos los problemas relacionados con las variables y sus atributos, apoyándose en los análisis anteriores.

- La DIRECCIÓN de memoria en la que se va a ubicar o está ubicado dicho objeto. Esta es una característica importante de los objetos, aunque pertenece a la fase de síntesis.
- Información sobre cuando apareció dicho objeto por primera vez en el programa (número de LÍNEA, y opcionalmente COLUMNA). De esta forma, en caso de tener que informar al usuario de cualquier problema con dicho objeto, se podrá indicar donde se encuentra.
- También es frecuente que en los registros de los objetos se guarden ENLACES con otros objetos por motivos diversos. Es como guardar referencias cruzadas entre varios objetos que estén relacionados; un poco más adelante se verá un ejemplo.

Además de estos campos, pueden existir muchos otros, dependiendo sobre todo de qué lenguaje de programación se esté implementando. Para empezar, depende de cuantos tipos de objetos diferentes maneje dicho lenguaje y del número de características que se le atribuyan. Hay muchas características muy comunes en otros lenguajes y que no se contemplan en Letra como son las siguientes:

- Un subtipo o formato de variables: en la práctica totalidad de los lenguajes de programación no basta con saber que "X" es una variable, sino que hace falta además, saber si es un entero, corto, largo, número flotante de precisión fija, de precisión doble, lógico (o "boolean"), de tipo carácter, un puntero, etc. En Letra, este problema no existe, pues todos los datos son del mismo tipo, aunque se puedan tratar indistintamente como si fueran enteros, lógicos o de carácter.
- No sólo se guardan en la tabla de objetos variables y etiquetas, sino que es muy frecuente que se guarden definiciones de tipos. Por ejemplo, si en el lenguaje C se define un nuevo tipo de datos como "byte" y se dice que es un "unsigned char", en la tabla de objetos se debe registrar este nuevo tipo. De hecho, se suele tener, además de la tabla de objetos, una tabla de tipos. Si un dato es de tipo puntero, se puede indicar en su registro un enlace que indique cual es el tipo al que apunta. En Letra ni hay punteros, ni se pueden definir nuevos tipos.

En la tabla de objetos se puede guardar información muy diversa, dependiendo del lenguaje

- Para guardar una definición de un registro ("record" de Pascal o "struct" de C) se pueden guardar todos los campos como objetos independientes y después, al registrar el registro, ponerle un enlace con los campos que contiene.
- Las funciones no existen como tales en Letra (aunque dispone de un arcaico sistema *GOSUB/RETURN* para simularlas), pero en el resto de los lenguajes, una función puede tener un valor de retorno con un tipo determinado y un número determinado de parámetros con sus correspondientes atributos.
- Además, en Letra se está obviando otro de los pilares fundamentales en los lenguajes de programación de alto nivel; los datos locales. En otros lenguajes, en la tabla de objetos se registran datos que, por ser locales, tienen sentido únicamente dentro de la función en la que se han definido, por lo que se puede incluso eliminar sus registros de la tabla de objetos una vez compilada la función.

Si se pretende construir un compilador para un lenguaje con algunas de las características que aquí se han descrito, es de vital importancia diseñar una "tabla de objetos" apropiada para registrar correctamente todos elementos del lenguaje.

En la tabla de objetos de Letra se guardará la información de datos, tablas y etiquetas que a continuación se detalla:

- *Tipo* (0-Sin tipo definido, 1-Dato, 2-Tabla y 3-Etiqueta).

LISTADO 2

```
Resumen de los objetos analizados en el programa

Ocupación de la tabla de objetos: 10 de 1024
Ocupación del vector de nombres:  63 de 12288

Listado de los objetos

<0>     Dato "fila"
        Hash 45, sin enlace
        Declarado=1, Usado=1
        Linea 12

<1>     Dato "columna"
        Hash 65, sin enlace
        Declarado=1, Usado=1
        Linea 14

<2>     Tabla de datos "ta"
        Hash 137, sin enlace
        Declarado=1, Usado=0
        Rango 0..1
        Linea 14

<3>     Tabla de datos "tb"
        Hash 138, sin enlace
        Declarado=1, Usado=1
        Rango 0..20
        Linea 15

<4>     Etiqueta ".funcion_b"
        Hash 236, sin enlace
        Declarado=1, Usado=1
        Linea 19

<5>     Etiqueta ".funcion_a"
        Hash 239, sin enlace
        Declarado=1, Usado=0
        Linea 23

<6>     Dato "dir_nave"
        Hash 141, sin enlace
        Declarado=1, Usado=1
        Linea 25

<7>     Dato "x_nave"
        Hash 141, enlace con <6>
        Declarado=1, Usado=1
        Linea 26

<8>     Tabla de datos "td"
        Hash 140, sin enlace
        Declarado=1, Usado=1
        Rango 0..2
        Linea 39

<9>     Tabla de datos "tc"
        Hash 139, sin enlace
        Declarado=1, Usado=1
        Rango 0..3
        Linea 41

Número de Datos    = 4
Número de Tablas   = 4
Número de Etiquetas = 2
```

Contenido de la tabla de objetos.

- *Nombre* (o identificador del objeto).
- *Enlace* (un enlace *hash* cuyo cometido se explica en el siguiente punto).
- *Declarado*, *Usado* (tal y como se detallaron antes).
- *Dirección* (campo no usado por ahora).

LISTADO 1

```
;-----------------------------------------------
;Programa para probar el analizador semántico
;-----------------------------------------------
;Se definen 4 datos, 4 tablas y 2 etiquetas
;Hay declarados/usados y usados/declarados
;Hay una tabla y una etiqueta que no se usan
;Hay dos datos con el mismo código hash
;-----------------------------------------------

datos

  fila columna

  ta[1]
  tb[20]=1:2:3:4

fin

  hacer .funcion_b

volver

.funcion_a

  si dir_nave=1
    x_nave=x_nave+1
  sino
    x_nave=x_nave-1
  fin

volver

.funcion_b

  fila=0
  mientras fila<tb[1]
    columna=0
    mientras columna<tb[2]
      columna=columna+td[1]
    fin
    fila=fila+tc[0]
  fin

volver

datos

  x_nave=20
  dir_nave=1

  tc[3]=1
  td[2]=falso:cierto

fin
```

Un ejemplo de programa compilado.

- *Longitud* (el rango de las tablas será de 0 a Longitud).
- *Línea* (primera aparición en el fuente de dicho objeto).

El campo *Longitud* sólo tiene sentido cuando el objeto sea de tipo tabla, pero se ha hecho una única estructura de registro común a los tres tipos con el fin de simplificar la codificación del compilador lo máximo posible. En el disco que acompaña a la revista se pueden encontrar los fuentes del compilador. Esta estructura está implementada en el fichero "letra.cpp".

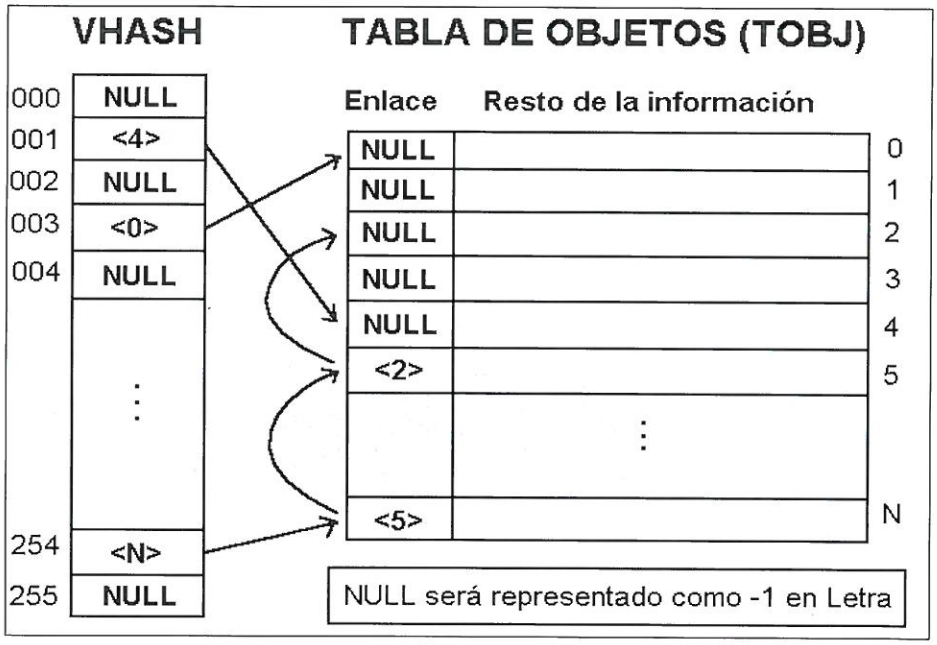


Figura 1: Mecanismo de búsqueda mediante función *HASH*

LA FUNCIÓN *HASH*

Para ver como se llega desde un identificador de dato escrito en un programa a su correspondiente registro en la tabla de objetos debe recordarse primero cómo funcionaba el analizador léxico al respecto.

El analizador léxico (o *scanner*) actuaba al encontrarse un identificador de dato o tabla de la misma forma; tras determinar que dicho identificador no era una de las palabras reservadas del lenguaje, informaba de que en el fuente había una pieza de tipo *p_identificador*, y la secuencia de caracteres que lo conformaban se pasaba en un vector denominado *nombre[ ]*. En el caso de que viniera una etiqueta (que reconocía porque comenzaba con un punto), indicaría que venia una pieza *p_etiqueta* y la secuencia de caracteres que la conformara (incluyendo el punto), se pasaba igualmente en *nombre[ ]*.

No es muy difícil suponer que, si cada vez que el compilador se encuentra un identificador, debe recorrer toda la tabla de objetos comprobando todos los nombres, uno por uno, para ver si el objeto es conocido, será sin duda un proceso realmente lento.

Para evitar en lo posible búsquedas demasiado lentas, se implementa el siguiente mecanismo:

- Se crea una función *hash* que, a partir de un identificador, obtiene un código numérico rápidamente. Dicho código numérico estará en un rango determinado (0..255 en el ejemplo). El código puede obtenerse con cualquier criterio, como por ejemplo, sumar todos los valores ASCII de los caracteres del identificador y quedarse con el valor truncado dentro del rango.
- Se crearán, no una sino, muchas listas de registros (256 en el ejemplo), de forma que cada identificador se insertará en una de ellas, en función de cual sea su valor *hash*.
- Cuando se quiera buscar un registro con un determinado identificador, primero se calculará su valor *hash*, para saber en qué lista buscar y, por tanto, se comparará únicamente con registros cuyos identificadores hayan obtenido el mismo valor.

El proceso de búsqueda de un identificador se acelera mediante la técnica hash

En la figura 1 puede observarse un esquema que representa este proceso. Primero habrá un vector *hash* (*VHASH*) de 256 posiciones. En cada posición se indica cual es el último objeto insertado en la tabla de objetos que obtuvo dicho valor *hash*. Si hay más de un objeto que ha obtenido dicho valor, el último insertado indicará en el campo *ENLACE* de su registro cual es el anterior registro con un identificador con su mismo valor *hash*.

Luego para buscar un objeto, primero se obtiene su valor *hash* a partir de su identificador (supóngase que es 238). Entonces se mira el contenido del *VHASH [238]*. Si éste no es -1 (lo que indicaría que no hay ningún objeto con dicho valor *hash*) será el número de un objeto de la tabla (supóngase ahora que es el objeto número 43). Se mira si el identificador coincide con el buscado, y si no coincide se mira si el objeto tiene un enlace con otro. Si lo tiene, por ejemplo con el objeto 22, es por que el identificador del objeto 22 también obtuvo el mismo valor de la función *hash*. Así se continuaría el proceso de búsqueda, que puede concluir tanto con la localización del objeto como con la inserción del mismo en caso de que no existiera.

Por tanto, cuando se inserta un nuevo objeto en la tabla, se debe enlazar con el resto de los objetos de su mismo valor *hash* que existieran en la tabla. Si los hubiera, bastaría con enlazarlo con el último, pues éste estará ya enlazado con los anteriores.

En el compilador que aquí se construye, cada vez que llega un identificador se compara con todas las palabras reservadas para ver si es o no una de ellas; un compilador comercial emplearía también un mecanismo *hash* para acelerar este proceso, aunque aquí no se haga.

EL VECTOR DE NOMBRES

De todos los campos de los registros de la tabla de objetos, el único que tiene una longitud variable es el del nombre del objeto. Esto representa un inconveniente pues, si se reservan por ejemplo 32 caracteres para dicho campo, se estaría limitando la longitud de los identificadores y además se inutiliza mucha memoria, pues la mayoría de los identificadores estarán, de media, entorno a los 6 caracteres.

Por ello no se guardarán los nombres en la tabla de objetos, sino en un vector de nombres denominado *VNOM[ ]*. Éste tendrá una capacidad determinada (en el ejemplo 12k) y en él se irán almacenando uno tras otro todos los identificadores que aparezcan en el programa (en forma ASCIIZ, es decir, acabados en un carácter nulo).

De esta forma, en la tabla de objetos simplemente se indicará en un entero la posición a partir de la cual se encuentra el identificador del objeto en el vector de nombres.

identificador y se busca si ya existía en la tabla de objetos.

- En caso afirmativo, se comprueba que el objeto no estuviera ya declarado (pues se trataría de una declaración duplicada) y que el tipo con el que se usó el objeto coincide con el tipo con el que se está declarando (para que no se defina como un dato y se use como si fuera una tabla o viceversa).
- En caso de que el objeto sea nuevo, se añade un nuevo registro a la tabla de objetos, se indica el tipo de objeto del que se trata, se mete su identificador en el vector de nombres, se define su enlace *hash* y se anotan otras características como indicar que está declarado, que no está usado y el número de línea en el que aparece.

Tras el análisis semántico el lenguaje ya está implementado con todas sus reglas

EL ACCESO A UN OBJETO

Durante el análisis semántico se realizan básicamente dos tareas:

- La primera, guardar el registro en las estructuras correspondientes de todos los objetos que aparecen declarados en el programa.
- La segunda, consultar en las mismas estructuras cada vez que se usa un objeto que se haga correctamente, de acuerdo con las reglas semánticas del lenguaje.

En algunos lenguajes, estos dos pasos han de ir en este orden, es decir, se deben declarar todos los objetos antes de poder ser usados; en otros como Letra, estos pasos se pueden dar en cualquier orden. Esto quiere decir que nuestro compilador creará el registro de un objeto nuevo tanto si se está declarando como si se está usando directamente. En este último caso, cuando posteriormente se declare el objeto, se completará su registro.

El proceso que se sigue para declarar un objeto es el siguiente:

- Primero se calcula el valor *hash* del

Otros errores típicos a comprobar son, por ejemplo, ver que no se desborde la capacidad de la tabla de objetos, ni la del vector de nombres. Si un compilador se queda sin espacio para registrar objetos no podrá en ningún caso continuar con el proceso de compilación.

El proceso que se sigue para usar un objeto es similar al de cuando se declara, pues también se crea el registro, en caso de que no existiera, se emite un error en caso de que existiera y tuviera un tipo diferente al que ahora se está utilizando, etc. Debe recordarse que en Letra es muy fácil averiguar el tipo de un objeto cuando se esta usando: si su identificador comienza con un punto es una etiqueta, en caso contrario, si a continuación de su identificador viene una pieza *p_abrir_corchete* se trata de una tabla y cuando viene cualquier otra se trata de un dato.

En la implementación que acompaña a este artículo, el analizador semántico proporciona básicamente los dos servicios siguientes:

int declarar("nombre",tipo);
int usar("nombre",tipo);

Ambos son invocados con el identificador del objeto que se está declarando y su tipo (un tipo enumerado donde dato=1, tabla=2 y etiqueta=3) y devuelven el número de registro de la tabla de objetos donde estaba o se ha situado ahora el objeto.

El primero será llamado únicamente en tres puntos del programa: al analizar una etiqueta, o una declaración de un dato o una declaración de una tabla. Sólo hay un detalle a resaltar y es que, cuando se declara un objeto de tipo tabla, se contarán después el número de elementos que contiene para indicarlo así en su registro. Una tabla se podía declarar de una de estas tres formas:

mi_tab[3]
mi_tab[3] = 0:1:2:3
mi_tab[ ] = 0:1:2:3

Recuerde que en Letra una tabla definida como *mi_tab[3]* tiene CUATRO posiciones en el rango [0..3]. Si no se define explícitamente la longitud de la tabla, se toma su longitud de la inicialización. Si se define su longitud de forma explícita y además se inicializa (como en el segundo caso) se producirá un error semántico cuando se inicialice con más valores de los que caben en la tabla.

El segundo procedimiento de análisis semántico (*usar()*) será invocado cuando se utilice una etiqueta (durante el análisis de la sentencia *HACER* y de la sentencia *IR*) y cuando se esté analizando un *<DATO>* (la regla de sintaxis en notación BNF que se implemento en un procedimiento llamado *an_dato()*) se llamará para "declarar el uso" de una tabla o dato según corresponda.

IDENTIFICADORES IDÉNTICOS

Letra no permite tener dos objetos con el mismo identificador, pero otros lenguajes si lo permiten. Por ejemplo, tener una variable local con el mismo nombre que tiene una variable global. En dicho caso, dentro de la parte del programa donde esté definida la variable local no se podrá acceder a la global. Supóngase la siguiente estructura para una tabla de objetos de otro lenguaje.

<*0*> Variable "xyz", Declarada, No usada, ...

Los campos de longitud variable representan un problema de cara a su almacenamiento

<*1*> Tabla "Churri", Declarada, 32 pos, ...
etc.

Si ahora se analiza una función donde exista una variable local denominada "xyz", la tabla de objetos tendrá durante su análisis el siguiente registro para dicho objeto:

El compilador registrará en sus estructuras de datos internas todos los objetos del programa

<*N*> Variable local "xyz", Declarada, ...

Cuando desde esa parte del código se busque una variable denominada "xyz", se encontrará primero (y únicamente) la variable local (pues la búsqueda se hacía desde el final hacia arriba), no pudiendo volver a ser accedida la variable "xyz" global hasta que, tras analizar la función, se elimine de la tabla de objetos el registro de la variable "xyz" local.

Hay más ocasiones en las que se podría tener varios objetos con el mismo nombre. Por ejemplo, en Letra se podría perfectamente permitir que hubieran tablas y datos con el mismo nombre:

<*0*> Dato "ab", declarado, ...
...
<*8*> Tabla "ab", declarada, ...

Sólo se tendría que modificar en el compilador que cuando buscara un objeto por la tabla de objetos exigiera que, además de coincidir el identificador con el buscado, coincidiera también el tipo. En la implementación actual sólo se exige que coincida el identificador, y si después el tipo no lo hace se produce un error.

LOS ERRORES SEMÁNTICOS

Hay dos tipos de errores semánticos: los primeros son aquellos que se detectan duraste el análisis del programa y los segundos son los que se detectan después.

Por ejemplo, si cuando se está compilando el programa se hace referencia a un objeto denominado "x" en algunas ocasiones como si fuera un dato y en otras como si fuera una tabla, se producirá un error durante el análisis del programa que, actualmente detiene definitivamente el proceso de compilación.

Sin embargo, si un objeto se usa una o varias veces durante todo el programa analizado pero no se declara en ninguna parte, el error no puede ser detectado hasta que no se haya acabado de analizar el programa, pues hasta el final se supone que podría venir siempre una declaración de éste.

Lo que se hace es una comprobación de coherencia de la tabla de símbolos tras el análisis del programa y, si un objeto esta marcado como usado pero no como declarado, se emite un error. Sin embargo, no se está comprobando actualmente la situación contraria, es decir, si un objeto se ha declarado pero no ha sido usado. Esta situación es ignorada en la implementación de Letra.

Hay un total de 6 errores semánticos que ahora se están detectando. Sobre ellos se ha hablado ya en este artículo. Estos aparecen reflejados en

el fuente del compilador que se entrega en el disco. En dicho compilador aparecen reflejados también los diferentes errores léxicos que se pueden producir; hasta el momento se advertía al usuario si se producía un error léxico, pero no se le indicaba de que tipo.

El analizador semántico accede a la tabla de objetos cuando se declara un objeto y cuando se usa

A modo de ilustración, pueden observarse los listados 1 y 2 que acompañan a este artículo: en el primero se muestra un pequeño programa escrito en Letra, que cuando se compila produce el listado 2, el cual es una muestra del contenido de la tabla de objetos tras analizar el programa.

En dicha tabla se pueden observar como quedan registrados los objetos que van apareciendo en el programa. Hay varios objetos que primero se declaran y después se usan y otros que se usan primero. Como puede observarse, esto no influye en absoluto en la tabla de objetos. De igual forma se pueden ver los valores *hash* que han obtenido los distintos identificadores, y como hay dos "x_nave" y "dir_nave" que como han obtenido el mismo valor *hash* se encuentran enlazados (el último con el primero). El programa ha compilado correctamente, por lo que han sido declarados todos los objetos, pero hay dos de ellos que no se usan en el programa, y así queda reflejado en la tabla de objetos de Letra.

RECUPERACIÓN DE SITUACIONES DE ERROR

El compilador que aquí se está desarrollando abortará su ejecución en cuanto se encuentre un error en el programa, sea éste del tipo que sea. Esto tiene el inconveniente de que, si el programa tiene varios errores, sólo se detectará el primero, y hasta que éste no sea corregido por el usuario y se vuelva a compilar el programa, no se detectará el siguiente. Si se quisiera que el compilador no se detuviera, se tendría que hacer lo siguiente:

- Para empezar, modificar la función de error para que tras emitir el error no abortara el programa, sino que llamara a una función de recuperación de errores y entonces retornara a la función llamante.
- Y en segundo lugar, construir una función de recuperación de errores. Dicha función tiene que tratar todos los errores que se pueden producir de forma individual, y para cada uno hacer "algún apaño a medida".

Por ejemplo, estas podrían ser algunas soluciones a las siguientes situaciones de error:

- Error léxico, literal sin cerrar: Se supone que el literal acaba justo cuando finaliza la línea del fuente en la que aparece, luego se "autocierra" el literal y se retorna.
- Error sintáctico, se esperaba "cerrar corchete" en lugar de "igual": se supone que en una tabla, tras el índice, se ha olvidado cerrar el corchete. Éste se "autoañade" y se retorna.
- Error semántico, dato no declarado: Puede "auto-declararse" dicho dato e inicializarlo a cero para continuar con la compilación.
- Etc.

Como puede observarse, no es nada fácil resolver algunos errores, y de hecho, hay muchos que no se pueden subsanar. Por ejemplo, si se olvida declarar una tabla, el compilador no puede adivinar cual era su longitud. Además, existe el problema añadido de que, al hacer alguna suposición, siempre se cometen fallos y el "apañar un error" suele desembocar en otros muchos errores.

INTRODUCCIÓN A LA GENERACIÓN DE CÓDIGO

El compilador ya está listo para comenzar a generar código. Cuando se comenzó a construir el compilador, se dijo que lo primero era aprender bien el lenguaje que se pretendía compilar. En esta ocasión, se puede decir que lo primero será conocer bien el lenguaje en el que se va a generar el código.

Ese lenguaje va a ser el "código máquina" (las instrucciones del ensamblador de la familia Intel 80x86), pero el lector puede estar tranquilo, pues no le será necesario tener conocimientos sobre dicho lenguaje para poder entender todo lo que se haga. No obstante, la meta será producir un fichero ejecutable a partir de un fuente en Letra, y dicho ejecutable estará construido en código máquina.

Pero aquí no se hablará de código máquina, sino de un código intermedio similar al código intermedio EM que generan otros compiladores, mucho más fácil de comprender y de manejar. Después se indicará la forma en la que este código se puede traducir fácilmente al código máquina.

El "modus operandi" será el siguiente: al igual que se ha hecho con el análisis semántico, se irán añadiendo sobre la estructura del analizador sintáctico llamadas a diversas funciones para ir generando el código.

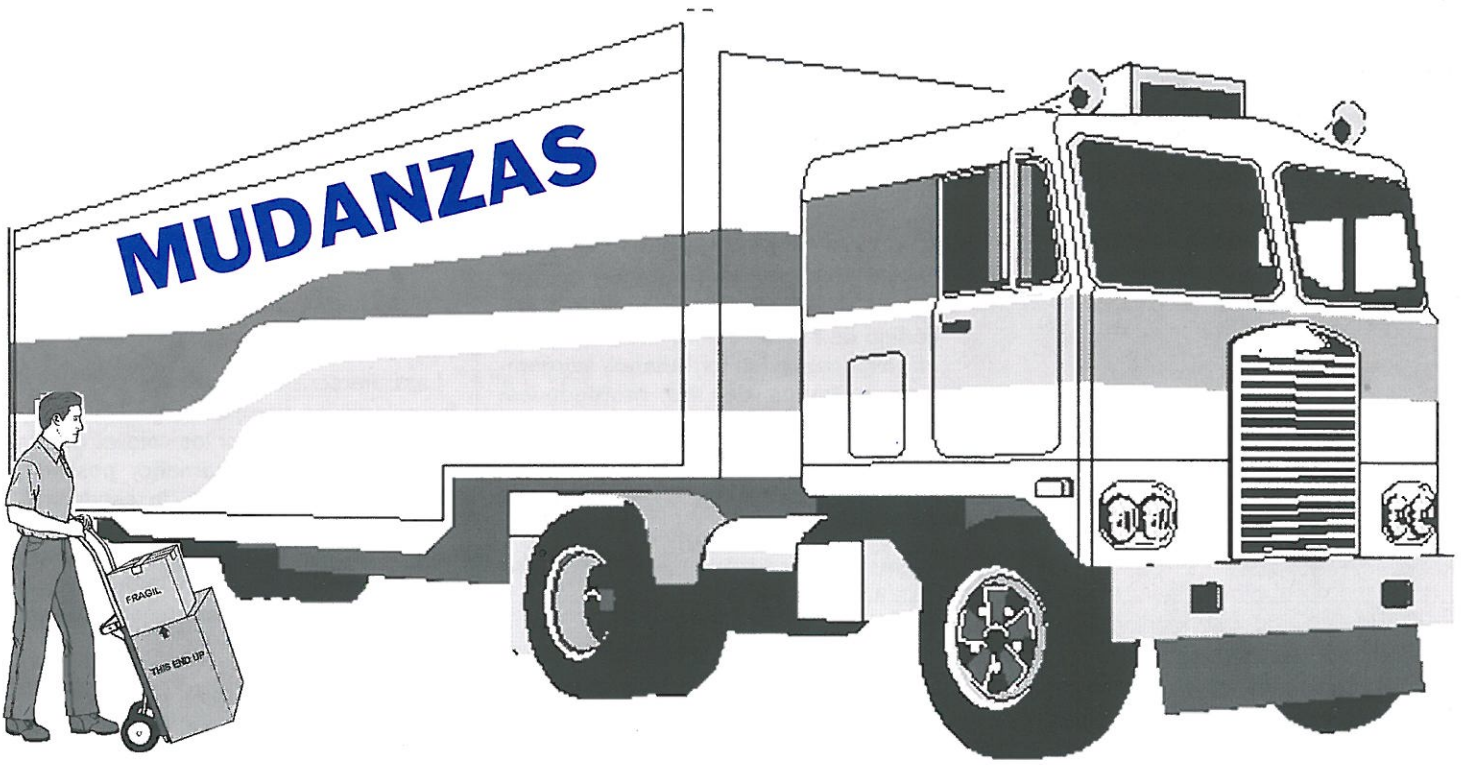
PRÓXIMO NÚMERO

Ya se ha acabado la fase de análisis, y en el siguiente artículo ya se estará generando código. La primera meta será definir el modelo de memoria en el que van a trabajar los programas compilados y generar los datos dentro de dicho modelo (generar las tablas y datos, asignándoles direcciones físicas en memoria e inicializándolas). Entonces se pasará a describir un lenguaje intermedio, denominado *EML* (se le ha añadido la "L" de lenguaje favorito) y que servirá para hacer realidad lo que hasta ahora podía parecer un sueño.

EN SÓLO PROGRAMADORES

Revista práctica para usuarios de PC

¡¡HEMOS CRECIDO!!

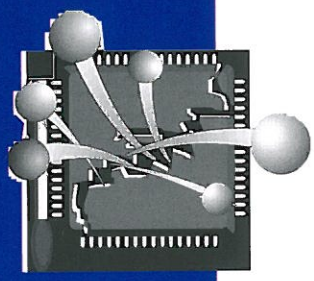


Y POR ESO, NOS MUDAMOS

c/ Aragoneses, 7
28100 Pol. Ind. Alcobendas. Madrid
Tel.: (91) 661 42 11 • Fax: (91) 661 43 86

LOS FORMULARIOS

Juan Manuel y Luis Martín



Los formularios son el soporte básico de las aplicaciones realizadas con Visual Basic. Cada una de las ventanas que componen la aplicación será diseñada a partir de un formulario. Sobre éstos se van situando los distintos controles que permitirán al usuario interactuar con la aplicación.

La primera fase en la realización de una aplicación con Visual Basic consiste en el diseño del interfáz de ésta con el usuario. Este proceso se realiza mediante la configuración de unas ventanas especiales denominadas *Formularios*, y que darán origen a las distintas ventanas y cajas de diálogo de que constará la aplicación. Además, los formularios son el soporte sobre el que se iran dibujando los distintos controles.

Las aplicaciones desarrolladas con Visual Basic pueden contener múltiples formularios. La responsabilidad de ir cargándolos y visualizándolos corresponde al programador a través del código de la aplicación.

El aspecto de los formularios se manipula a través de sus propiedades. Además, disponen de un conjunto de métodos que permiten realizar determinadas operaciones con ellos. Finalmente, disponen también de un cierto repertorio de procedimientos de evento que les permiten responder a los distintos eventos producidos.

EL FORMULARIO POR DEFECTO

Al comenzar un nuevo proyecto, Visual Basic incluye un primer formulario por defecto, denominado *Form1*. Se trata de un prototipo de ventana de Windows con todos los elementos típicos de éstas (Barra de Título, Botones "Maximizar" y "Minimizar", etc). Además, en el área de cliente se visualiza una rejilla de puntos que permitirá alinear los controles que se sitúen sobre él. La figura 1 muestra el aspecto y los elementos que componen el formulario.

Con este formulario pueden realizarse todas las operaciones que normalmente se realizan con las ventanas (maximizarlo, moverlo, cambiar su tamaño, etc). Si se cierra, desaparece de la pantalla, aunque no de la memoria. Para recuperarlo será necesario seleccionar el módulo *Form1* en la Ventana de Proyecto, y hacer clic sobre el botón *Ver Formulario*.

Si se ejecuta el proyecto *Project1* sin hacer cambios sobre *Form1*, se visualizará una ventana idéntica al formulario, aunque sin rejilla. Sobre ella también es posible realizar las operaciones típicas anteriormente descritas. Para ejecutar el proyecto bastará con seleccionar la opción *Iniciar* del menú *Ejecutar* (o pulsar [F5]). Para detener la ejecución, bastará con seleccionar la opción *Terminar* del menú *Ejecutar* (o cerrar la ventana).

ESPECIFICACIÓN DE LAS PROPIEDADES

Las características que presentarán los formularios en tiempo de ejecución vienen determinadas por los valores de sus propiedades (color, tamaño, posición, etc). En fase de diseño, la especificación de los valores de las propiedades, tanto para los formularios como para los controles, se realiza en la Ventana de Propiedades, cuyo aspecto se muestra en la figura 2.

Para acceder a la Ventana de Propiedades, bastará con seleccionar la opción *Propiedades* del menú *Ver* (o pulsar [F4]). Dentro de ella, la lista desplegable de la parte superior contiene los nombres de todos los objetos del formulario actual, incluido el propio formulario. La columna de la izquierda contiene los nombres de las propiedades del objeto seleccionado, mientras que la de la derecha contiene los valores correspondientes.

Para especificar el valor de una propiedad deben seguirse los siguientes pasos:

1. Seleccionar el formulario o control que contiene la propiedad.
2. Acceder a la ventana de propiedades.

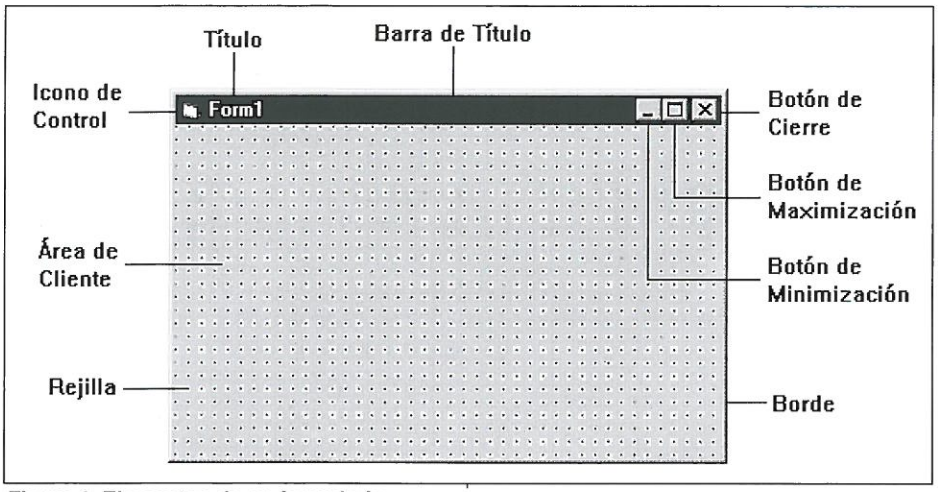


Figura 1. Elementos de un formulario.

3. Seleccionar el nombre de la propiedad que se desea especificar.
4. Escribir el nuevo valor de la propiedad.

Si se trata de una propiedad con un rango de valores limitado, bastará con pulsar el botón que aparece a su derecha y seleccionar el valor en la lista desplegada. En este caso, también es posible ir cambiando de un valor al siguiente haciendo doble clic sobre el nombre de la propiedad.

También es posible alterar el valor de las propiedades de los objetos en tiempo de ejecución. Esto se realiza mediante sentencias de asignación, dentro del código de los procedimientos. La sintaxis general para la asignación de valores a las propiedades en el código es la siguiente:

Objeto.Propiedad = Valor

Así, por ejemplo, para cambiar la propiedad *Top* del formulario *Form1* puede utilizarse la siguiente instrucción:

Form1.Top = 2450

Visual Basic también dispone de una serie de constantes intrínsecas que representan los valores de algunas de las propiedades con un repertorio limitado. La utilización de estas contantes en lugar de los valores que representan facilitan la comprensión del código, y resultan más fáciles de recordar para el programador. Estas constantes comienzan con el prefijo *vb* y se encuentran definidas en la librería de objetos de Visual Basic, por lo que pueden ser utilizadas en cualquier parte del código. Para conocer las constantes intrínsecas de una propiedad puede consultarse la ayuda *On-Line* del propio Visual Basic.

Form1.MousePointer = vbHourglass

EL NOMBRE Y EL TÍTULO DEL FORMULARIO

En Visual Basic, los objetos (formularios o controles) se identifican por un nombre. Este nombre también será el que se utilice para referenciarlos en el código. El nombre de un objeto siempre reside en su propiedad *Name*. El nombre puede contener hasta 40 caracteres, debiendo ser el primero una letra. No se admiten espacios en blanco ni caracteres especiales, salvo el subrayado (_). Al crear objetos, Visual Basic les asigna un nombre por defecto, dependiendo del tipo de objeto y un número secuencial. Así, por ejemplo, para el caso de los formularios, los nombres asignados serán *Form1*, *Form2*, y así sucesivamente.

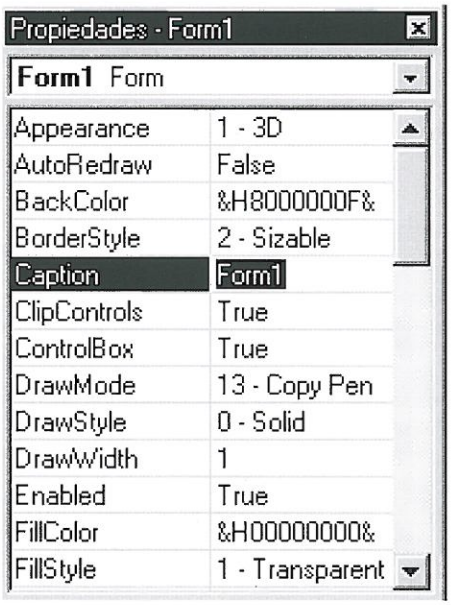


Figura 2. ventana de propiedades.

Sin embargo, para nombrar los formularios es conveniente utilizar un nombre indicativo de la misión realizada por el mismo. Además, para distinguir el nombre de un formulario del de otros objetos suele utilizarse el prefijo *frm*. Así, por ejemplo, pueden utilizarse nombres como *frmInicial*, *frmLeer*, *frmSalir*, etc.

Por otra parte, el texto que se visualiza en la Barra de Título del formulario puede ser cualquiera. Aunque inicialmente coincide con el nombre del formulario, dicho texto reside en la propiedad *Caption*. Estas propiedades no deben ser confundidas, ya que el valor de la propiedad *Caption* es meramente descriptivo de cara al usuario.

Cada uno de los formularios que intervienen en un proyecto se almacena en un archivo independiente con la extensión *.FRM*. Todos los archivos de formulario se salvarán cada vez que se salve el proyecto, aunque es posible salvar cada uno de ellos independientemente. Para ello, debe seleccionarse el formulario en la ventana de proyecto y, seguidamente, seleccionar la opción *Guardar Archivo* o *Guardar Archivo Como* en el menú *Archivo*. Por defecto, el nombre utilizado para salvar el archivo coincide con la propiedad *Name*.

LA POSICIÓN Y EL TAMAÑO DEL FORMULARIO

La posición del formulario en la pantalla se especifica en las propiedades *Left* y *Top*, que indican la distancia entre la esquina superior izquierda del formulario y la de la pantalla. Igualmente, el tamaño del formulario se especifica en las propiedades *Width* y *Height*, que indican la anchura y altura del mismo, respectivamente. Estas cuatro propiedades vienen expresadas en una unidad de medida especial, independiente del sistema de video, denominada *Twip* (en una pulgada entran aproximadamente 1440 *twips*).

Sin embargo, la posición y el tamaño del formulario también puede alterarse mediante operaciones de arrastre con el ratón, como en cualquier ventana de Windows. Los cambios que así se realicen quedarán automáticamente reflejados en las propiedades.

EL ESTILO DEL FORMULARIO

Para especificar el tipo de borde del formulario en tiempo de ejecución se utiliza la propiedad *BorderStyle*. Sin embargo, esta propiedad también influye en su comportamiento, ya que determina si será posible moverlo, cambiar su tamaño, maximizarlo, etc. La tabla 1 muestra los valores posibles de esta propiedad, que sólo serán reflejados en tiempo de ejecución. Además, su valor únicamente podrá ser alterado en tiempo de diseño.

Existen otras tres propiedades que afectan al aspecto y comportamiento del formulario. Se trata de las propiedades *MinButton*, *MaxButton* y *ControlBox*, que determinan si el formulario dispondrá de botón "Minimizar", "Maximizar" y menú de control, respectivamente. Un valor *True* en cualquiera de estas propiedades indica que el formulario contendrá el elemento correspondiente, mientras que *False* indica lo contrario. Su acción combinada con *BorderStyle* determina la apariencia del formulario. La figura 3 muestra varios tipos de ventana combinando los valores de estas propiedades.

TABLA 1

VALOR	NOMBRE	DESCRIPCIÓN
0	None	Sin borde ni barra de título. El usuario no puede moverlo ni cambiar su tamaño.
1	Fixed Single	Borde de línea simple y barra de título con todos sus elementos. El usuario puede moverlo pero no cambiar su tamaño
2	Sizable	Borde normal y barra de título con todos sus elementos. El usuario puede moverlo y cambiar su tamaño
3	Fixed Double	Borde de línea gruesa y barra de título sin botones "Maximizar" y "Minimizar". El usuario puede moverlo pero no cambiar su tamaño
4	Fixed ToolWindow	Barra de título con tipo de letra reducido y botón "Cerrar". El usuario puede moverlo pero no cambiar su tamaño (sólo para Windows'95)
5	Sizable ToolWindow	Barra de título con tipo de letra reducido y botón "Cerrar". El usuario puede moverlo y cambiar su tamaño (sólo para Windows'95)

Valores de la propiedad BordeStyle.

ICONOS Y PUNTEROS DEL FORMULARIO

El icono representativo del formulario se especifica en la propiedad *Icon* del mismo. En Windows'95, este icono se visualiza en la barra de título, mientras que en Windows 3.X y Windows NT sólo es visible al minimizarlo. La especificación de esta propiedad en tiempo de diseño se realiza indicando el nombre de un archivo de icono (.ICO). Para ello, la ventana de propiedades dispone de un botón con tres puntos suspensivos a la derecha del valor. Al hacer clic sobre él se despliega un cuadro de diálogo que permite especificar el nombre y la ruta del archivo de icono. En tiempo de ejecución, la especificación de esta propiedad puede realizarse con la función *LoadPicture*.

Los formularios son el soporte básico de la Interfaz de la aplicación.

Form1.Icon = LoadPicture ("C:\ICONS\DISK01.ICO")

Visual Basic también permite cambiar el aspecto del puntero del ratón cuando éste se encuentra sobre el formulario. Para ello, debe utilizarse la propiedad *MousePointer*. La tabla 2 muestra los posibles valores de esta propiedad. De entre ellos, el valor 99 es especial, ya que permite especificar un puntero personalizado. Para ello, puede utilizarse un archivo de icono (.ICO) o de cursor (.CUR), que debe indicarse en la propiedad *MouseIcon*. La especificación de esta propiedad se realiza de forma idéntica a la propiedad *Icon*.

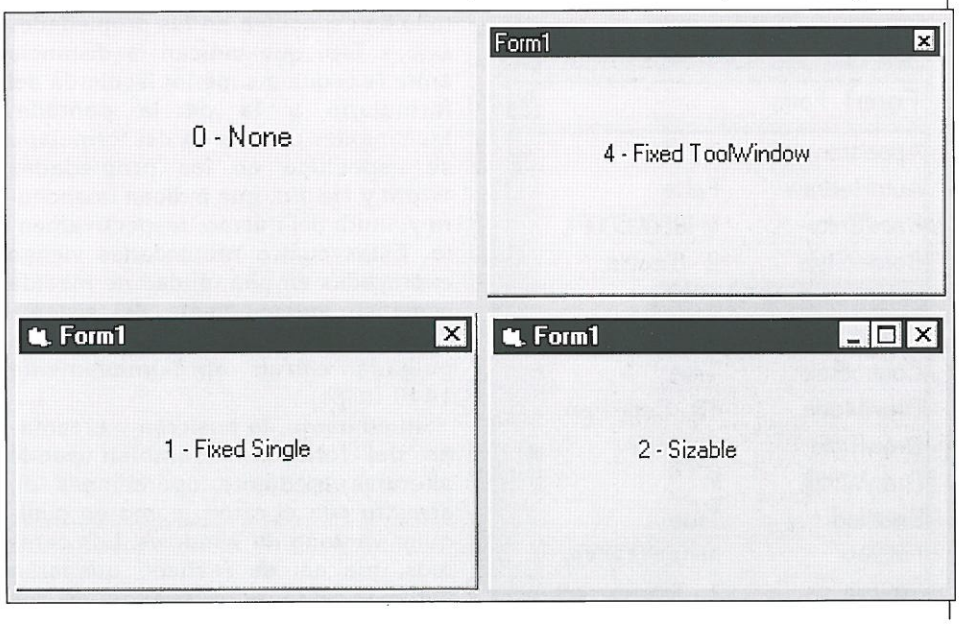


Figura 3. Varios tipos de formularios con sus correspondientes valores de borde estilo.

TABLA 2

VALOR	CONSTANTE	DESCRIPCIÓN
0	vbDefault	Puntero por defecto
1	vbArrow	Flecha
2	vbCrosshair	Cruz
3	vbIbeam	Barra vertical
4	vbIconPointer	Icono
5	vbSizePointer	Flecha cuádruple
6	vbSizeNESW	Flecha doble diagonal NE-SO
7	vbSizeNS	Flecha doble vertical
8	vbSizeNWSE	Flecha doble diagonal NO-SE
9	vbSizeWE	Flecha doble horizontal
10	vbUpArrow	Flecha arriba
11	vbHourglass	Reloj de arena
12	vbNoDrop	No soltar
99	vbCustom	Puntero personalizado

EL ESTADO DEL FORMULARIO

El estado de visualización del formulario en tiempo de ejecución se indica en la propiedad *WindowState*. Los valores 1 (*Minimized*) y 2 (*Maximized*) indican que el formulario se encuentra minimizado o maximizado, respectivamente. El valor por defecto es 0 (*Normal*), indicando que se visualiza normalmente.

También es posible indicar si el formulario se encuentra o no visible, utilizando para ello la propiedad *Visible*. Un valor *True* indica que el formulario se encuentra visible, mientras que *False* indica que se encuentra oculto. En este último caso, el formulario permanece cargado en memoria, aunque el usuario no puede interactuar con él.

En el caso de Windows'95, también es posible indicar si el formulario debe visualizarse como un botón en la Barra de Tareas (*Taskbar*). Para ello, se utiliza la propiedad *ShowInTaskbar*, cuyo valor por defecto es *True*.

LOS COLORES DEL FORMULARIO

Para especificar los colores de visualización de un formulario, se utilizan las propiedades *BackColor* y *ForeColor*. La primera indica el color de fondo del área de cliente, mientras que la segunda indica el color del texto y los gráficos que se visualizan sobre él. Estos colores serán los utilizados por los métodos de escritura de textos y dibujo de gráficos del formulario (*Print*, *Line*, *Circle*, etc).

En Visual Basic, cada color se representa por un número, generalmente expresado en hexadecimal (por ejemplo, *&H80000005&*). Este valor se obtiene mediante la combinación de tres magnitudes que representan los tres colores primarios (rojo, verde y azul). Sin embargo, en tiempo de diseño es posible seleccionar el valor en una paleta de colores. Para ello, la ventana de propiedades dispone de un botón con tres puntos suspensivos a la derecha del valor. Al hacer clic sobre él se despliega un cuadro de diálogo como el mostrado en la figura 4, permitiendo seleccionar un color. En tiempo de ejecución, la especificación de esta propiedad puede realizarse directamente, utilizando las constantes intrínsecas, o mediante las funciones *RGB* y *QBColor*, que serán analizadas en posteriores artículos.

Form1.BackColor = vbGreen

EL TIPO DE LETRA

El aspecto y tipo de letra utilizado para escribir sobre el formu-

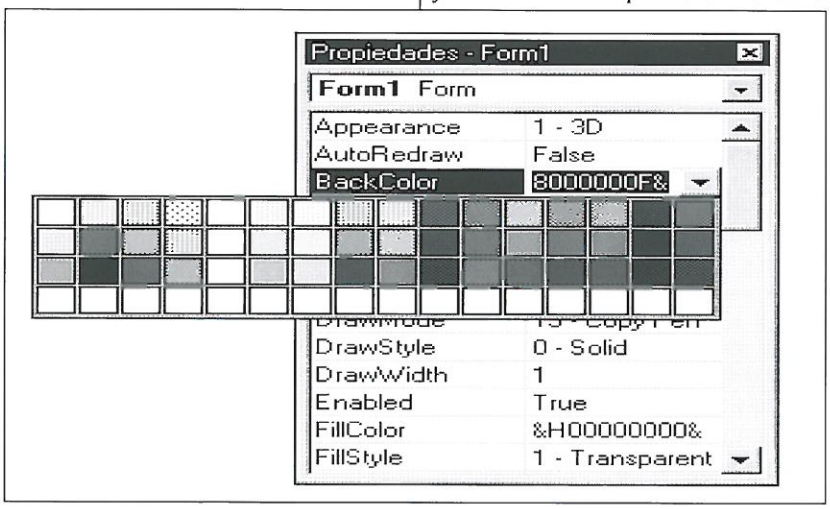


TABLA 3

PROPIEDAD	DESCRIPCIÓN
Negrita	Italic
Itálica	Itálica
Name	Nombre de la fuente
Size	Tamaño de la fuente
StrikeThrough	Tachada
Uderline	Subrayada
Weight	Grosor de la fuente

Propiedades del objeto Font, para la especificación de la fuente y sus características.

lario se indica en la propiedad *Font*. A diferencia del resto, esta propiedad es un objeto. Por tanto, contiene a su vez varias propiedades que indican el nombre de la fuente, el tamaño, negrita, cursiva, etc. La tabla 3 muestra las propiedades del objeto *Font*.

En tiempo de diseño, es posible indicar el valor de esta propiedad seleccionándolo en un cuadro de diálogo. Para ello, la ventana de propiedades dispone de un botón con tres puntos suspensivos a la derecha del valor. Al hacer clic sobre él se despliega un cuadro de diálogo como el mostrado en la figura 5, permitiendo seleccionar los valores de cada una de las propiedades del objeto *Font*.

En tiempo de ejecución, el valor debe especificarse directamente. Sin embargo, debe tenerse en cuenta que éste a su vez se compone de varias propiedades que deben ser especificadas individualmente. Para ello, se utiliza el mismo tipo de sintaxis, es decir, el nombre del objeto y la propiedad separados por un punto.

```
Form1.Font.Name = "COURIER"
Form1.Font.Size = 10
Form1.Font.Bold = True
```

El objeto *Font* es una novedad de la versión 4.0. En versiones anteriores, la especificación del tipo de la fuente y sus características se realizaba mediante un repertorio de propiedades del propio formulario (*FontBold*, *FontItalic*, *FontName*,*FontSize*, *FontStrikethru*, *FontTransparent* y *FontUnderline*). Estas propiedades pueden seguir utilizándose por compatibilidad, aunque su uso no es recomendable.

LAS PROPIEDADES DE ESCALA

Como ya se ha mencionado anteriormente, la posición del formulario dentro de la pantalla se especifica en una unidad denominada *twip*. Sin embargo, la posición de los distintos elementos que contiene el formulario (controles, textos, dibujos, etc) no viene referida a la pantalla, sino al propio formulario. La unidad de medida también es el *twip*, aunque en este caso puede cambiarse utilizando la propiedad *ScaleMode* del formulario. Esta propiedad permite utilizar distintos tipos de unidades de

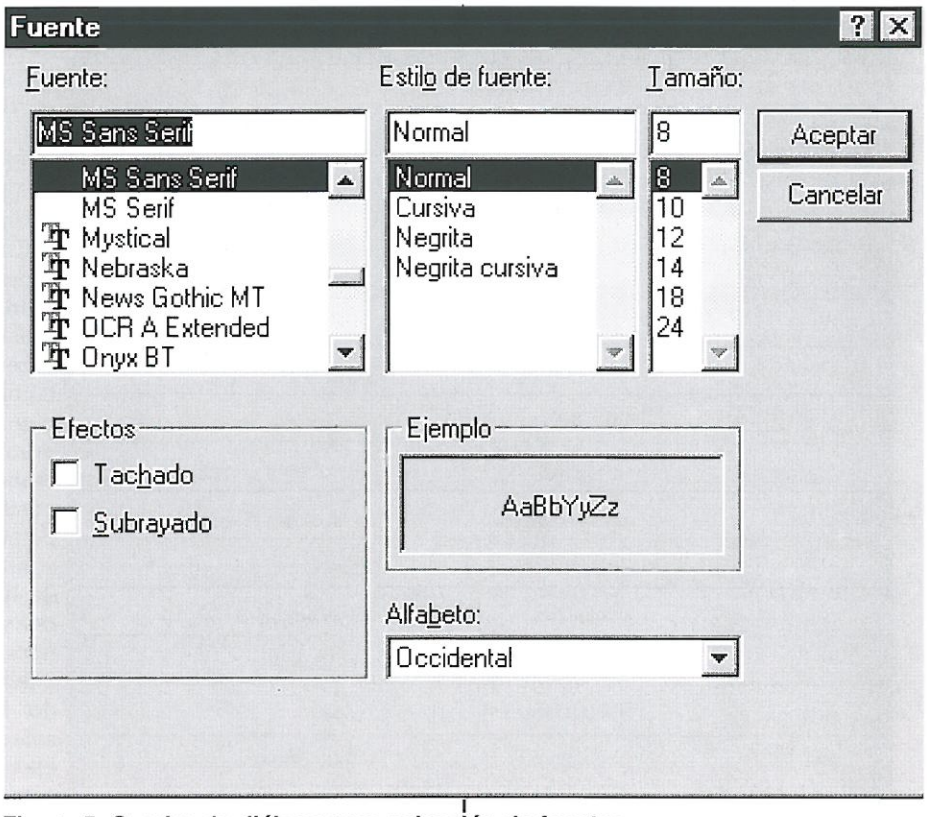


Figura 5. Cuadro de diálogo para selección de fuentes.

medida (*twips*, puntos, *pixels*, caracteres, pulgadas, milímetros o centímetros).

Existe además un valor especial de *ScaleMode* que permite definir una unidad de medida personalizada. Se trata del valor 0 (*Custom*) que debe utilizarse en combinación con las propiedades *ScaleHeight*, *ScaleLeft*, *ScaleTop* y *ScaleWidth*. Estas propiedades permiten definir un sistema de coordenadas personalizado dentro del formulario. Su funcionamiento será analizado en detalle en posteriores artículos.

OTRAS PROPIEDADES

Además de las propiedades que han sido descritas hasta ahora, los formularios disponen de muchas otras propiedades. La mayoría de ellas están relacionadas con temas específicos. Así, por ejemplo, las propiedades *CurrentX*, *DrawMode*, *FillStyle*, etc, están relacionadas con el dibujo de gráficos, las propiedades *LinkMode* y *LinkTopic* con el Intercambio Dinámico de Datos, etc.

Estas propiedades serán estudiadas en detalle al analizar los temas correspondientes.

Las propiedades definen el aspecto y comportamiento de los formularios.

CONCLUSIÓN

El correcto manejo de los formularios y de sus propiedades es fundamental en el desarrollo de la aplicación. Por ello, es conveniente practicar alterando los valores de las distintas propiedades que han sido analizadas y comprobar sus efectos en tiempo de ejecución. Para ello, puede utilizarse el formulario por defecto.

En el próximo artículo se analizarán los otros elementos que componen la interfáz de la aplicación con el usuario, es decir, los controles. Algunas de sus propiedades serán comunes a varias de las que han sido explicadas para los formularios.

LOS MODOS GRÁFICOS DE LA VGA (II)

Sergio Ríos

Tal y como vimos en anteriores artículos, en la arquitectura del adaptador gráfico VGA ocupan un lugar central los cuatro planos de bits, que no son más que una organización hardware específica de la memoria de vídeo.

Según esta organización de la memoria, cada uno de los planos de bits contiene uno de los bits del valor de los pixels mostrados en pantalla (recordemos que en los modos de 16 colores se utilizan cuatro bits por pixel), tal como ilustra la Figura 1a.

Obsérvese que en el primer byte de cada uno de los planos se guarda la información relativa a uno de los bits (una componente) de hasta 8 pixels contiguos.

Lo realmente importante es que toda la información relativa a estos 8 pixels contiguos es accesible desde una única dirección de memoria de la CPU; esto es, cuando la CPU hace una referencia a una dirección de la memoria de vídeo, se accede a un valor de 32 bits (8 pixels, con 4 bits por pixel).

Esto se debe a que el bus de datos interno de la VGA es de 32 bits, aunque la transferencia de datos con la CPU se realiza a través de un bus de 8 bits (en la VGA estándar). Para gestionar este intercambio heterogéneo de datos existe un subsistema específico en la arquitectura de la VGA: el controlador gráfico, del que nos ocuparemos más adelante.

Internamente, la VGA siempre trabaja con planos de bits, si bien en algunos modos gráficos esta organización es totalmente transparente para el programador, como es el caso del modo 13h, en el que aparentemente se utiliza un mecanismo de direccionamiento lineal de la memoria de video, y en el que cada pixel (un byte) posee su propia dirección de memoria.

Los modos gráficos de 16 colores utilizan de forma directa la organización en planos de bits, con lo que, en estos casos, la organización física de la memoria coincide con la estructura lógica observada por el programador. Es por ello que estos modos reciben la denominación de *modos nativos*.

La utilización de planos de bits tiene dos ventajas innegables. En primer lugar, se soluciona un serio problema: la limitación del espacio de direcciones de la CPU. En efecto, ahora es posible acceder mediante una dirección de la CPU a más de un byte de la memoria de vídeo.

En segundo lugar, se consigue un notable incremento de prestaciones, puesto que las operaciones internas son de 32 bits, y sobre todo, porque se pueden realizar varias operaciones de forma simultánea en los cuatro planos de bits.

Retomemos ahora el tema del controlador gráfico. Como decíamos antes, este subsistema es el encargado de gestionar el intercambio de datos entre la memoria de video y la CPU. Para ello, dispone de cuatro pequeños buffers de 1 byte denominados *latches* ("cerrojos") y de diferentes mecanismos de selección y transferencia de datos, tal como se puede apreciar en la Figura 1b.

Veamos cómo es, a grandes ras-

En este artículo, enmarcado en la programación avanzada del adaptador gráfico VGA, nos ocuparemos de la gestión de los modos gráficos nativos del mismo, es decir, los modos basados en planos de bits

gos, el intercambio de datos entre la memoria de video y la CPU:

Al realizar una lectura de la memoria de video, siempre se cargan los *latches* con el valor del byte direccionado en cada uno de los planos, con lo que mediante una operación de 8 bits conseguimos una transferencia efectiva de 32 bits. A continuación se selecciona uno de estos bytes (almacenado en uno de los *latches*) y se envía a la CPU, usando para ello uno de los diferentes modos de operación del controlador gráfico.

Cuando se realiza una escritura, el byte ofrecido por la CPU se puede combinar con el contenido de los *latches* (previamente cargados) y con el contenido de los valores de determinados registros del controlador gráfico; el resultado puede afectar a uno o varios planos. Los distintos modos de funcionamiento del controlador gráfico determinan el tipo de combinaciones realizadas con el dato ofrecido por la CPU.

Los modos de operación del controlador gráfico pueden ser de dos tipos: orientados a pixels y orientados a bytes. En los modos orientados a pixels, se considera el contenido de los *latches* como un conjunto de 8 pixels de 4 bits cada uno. En los modos orientados a bytes, se trabaja con el contenido de cada uno de los *latches* por separado (es decir, se trabaja con una de las componentes de los 8 pixels de forma individual). Algunas operaciones puntuales son una combinación de los dos mecanismos aquí descritos.

El controlador gráfico dispone de nueve registros, que permiten definir el modo de operación deseado, así como especificar los valores que intervienen en las distintas operaciones de transferencia de datos entre la memoria de video y la CPU.

Es posible acceder a los registros del controlador gráfico a través de dos puertos de E/S: 3CEh (índice) y 3CFh (datos). En la Tabla 1 se muestran los nueve registros, así como el valor que toman por defecto gracias a la BIOS.

Los modos de operación del controlador gráfico son seis: 2 modos de lectura y 4 modos de escritura. Estos

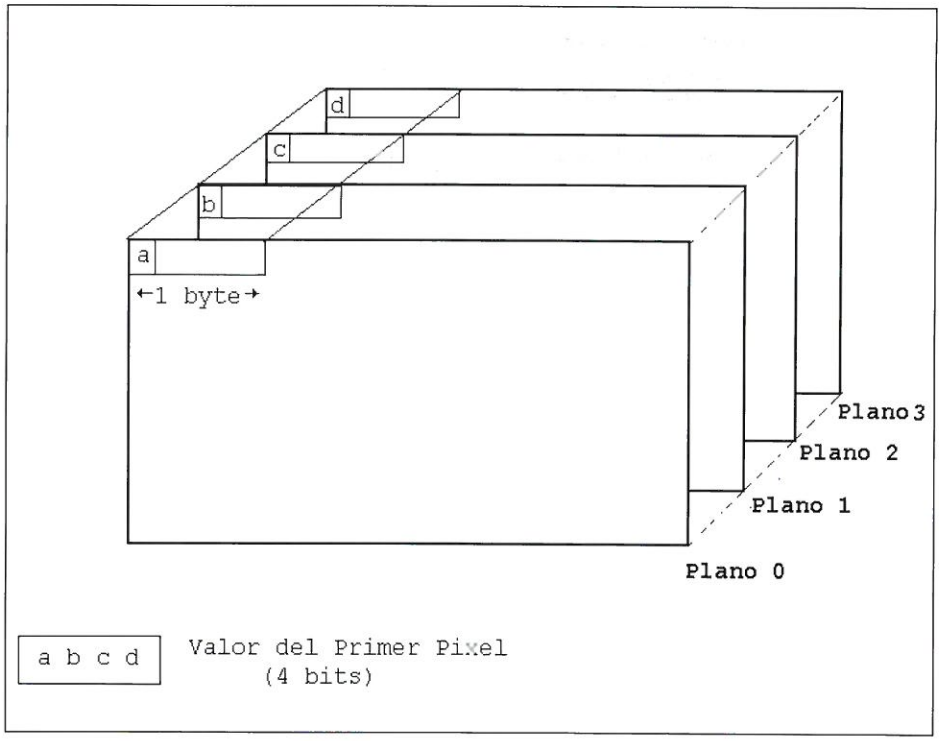


Figura 1.A.- Organización de la memoria de vídeo en planos de bits.

modos tratan de cubrir todas las necesidades que puedan surgir, y cada uno de ellos puede resultar especialmente indicado en determinadas circunstancias.

El *Registro de Modo* del controlador gráfico, como su propio nombre indica, se utiliza para especificar el modo de lectura o de escritura deseados. El bit 3 de este registro indica el modo de lectura, y los bits 0 y 1 indican el modo de escritura.

Veamos cada uno de los modos del controlador gráfico.

MODO 0 DE LECTURA

En el Modo 0 de lectura, cada vez

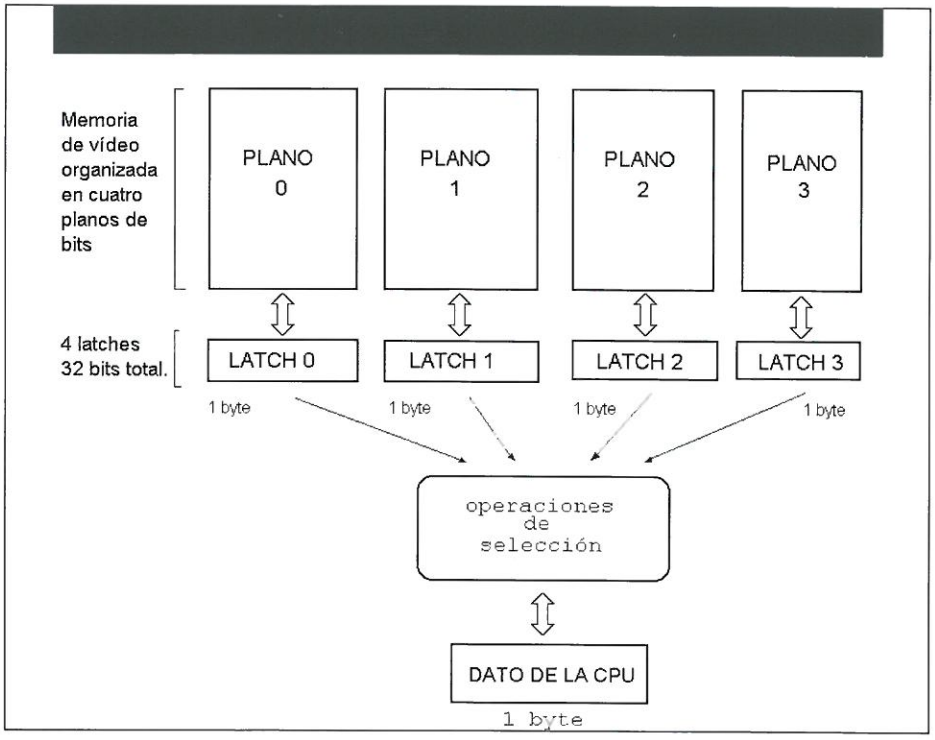


Figura 1 B. Intercambio de datos entre la CPU y la VGA.

TABLA 1

Registro	Nombre	Valor por defecto
00h	Set/Reset	0
01h	Enable Set/Reset	0
02h	Color Compare	0
03h	Data Rotate/Function Select	0
04h	Read Map Select	0
05h	Mode	x0h
06h	Misc	x
07h	Color Don't Care	0Fh
08h	Bit Mask	FFh

Registros del controlador de la VGA.

TABLA 2

Valor	Operación
00	Sustitución
01	AND
10	OR
11	XOR

Operaciones disponibles en las ALUs del controlador gráfico de la VGA.

que la CPU intenta leer un dato, se cargan los cuatro *latches* (como ocurre siempre que hay una lectura), y se envía a la CPU el valor de uno de ellos. El *latch* seleccionado será el especificado en los 2 bits de menor peso del registro *Read Map Select*, tal como ilustra la Figura 2.

La principal utilidad de este modo de operación está en las transferencias masivas de datos entre los planos de bits y la memoria RAM del sistema o archivos.

MODO 1 DE LECTURA

En el modo 1 de lectura, el controlador gráfico compara el valor indicado en los 4 bits de menor peso del registro *Color Compare* con el valor de cada uno de los 8 pixels cargados en los *latches*.

Tal como se puede observar en la Figura 3, el valor del byte devuelto a la CPU refleja el resultado de esta comparación para cada uno de los pixels. Un bit estará puesto a 1 si el valor indicado en el registro *Color Compare* coincide con el valor del pixel correspondiente.

En la Figura 3 también se puede apreciar el efecto del registro *Color Donít Care* (*CDC*). Antes de la comparación, se realiza dos *ANDs* lógicos: entre el registro *Color Compare* y el *CDC*, y entre el valor de cada pixel y el *CDC*. Esto se utiliza para excluir uno o varios *latches* de la comparación.

MODO 0 DE ESCRITURA

El modo 0 de escritura es bastante potente y versátil, lo cual redunda en una mayor complejidad. En este modo intervienen varios registros: *Enable Set/Reset*, *Set/Reset*, *Data Rotate/Function Select* y *Bit Mask*.

Este modo, las escrituras en la memoria de video se pueden realizar de dos formas, en función del valor del registro *Enable Set/Reset*.

En la Figura 4 se puede observar el funcionamiento de este modo cuando el registro *Enable Set/Reset* toma el valor 0Fh.

El funcionamiento es el siguiente: En el registro *Set/Reset* se especifica un valor que se utiliza en una operación en la que intervienen los 8 pixels previamente cargados en los *latches*. Estas operaciones se llevan a cabo en las distintas unidades aritmético-lógicas (*ALUs*) que para tal efecto existen en el controlador gráfico. El tipo de operación viene indicado en los bits 3 y 4 del registro *Data*

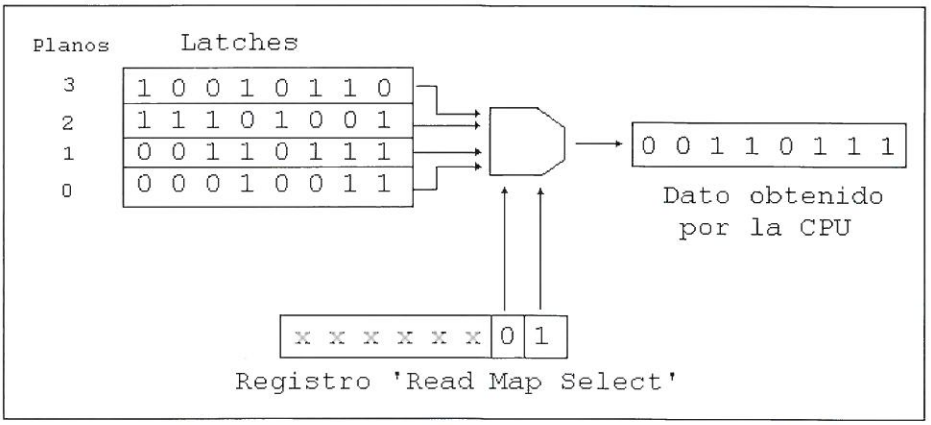

Planos
Latches
3
2
1
0
1 0 0 1 0 1 1 0
1 1 1 0 1 0 0 1
0 0 1 1 0 1 1 1
0 0 0 1 0 0 1 1
0 0 1 1 0 1 1 1
Dato obtenido por la CPU
x x x x x x 0 1
Registro 'Read Map Select'

Figura 2. Modo 0 de Lectura.

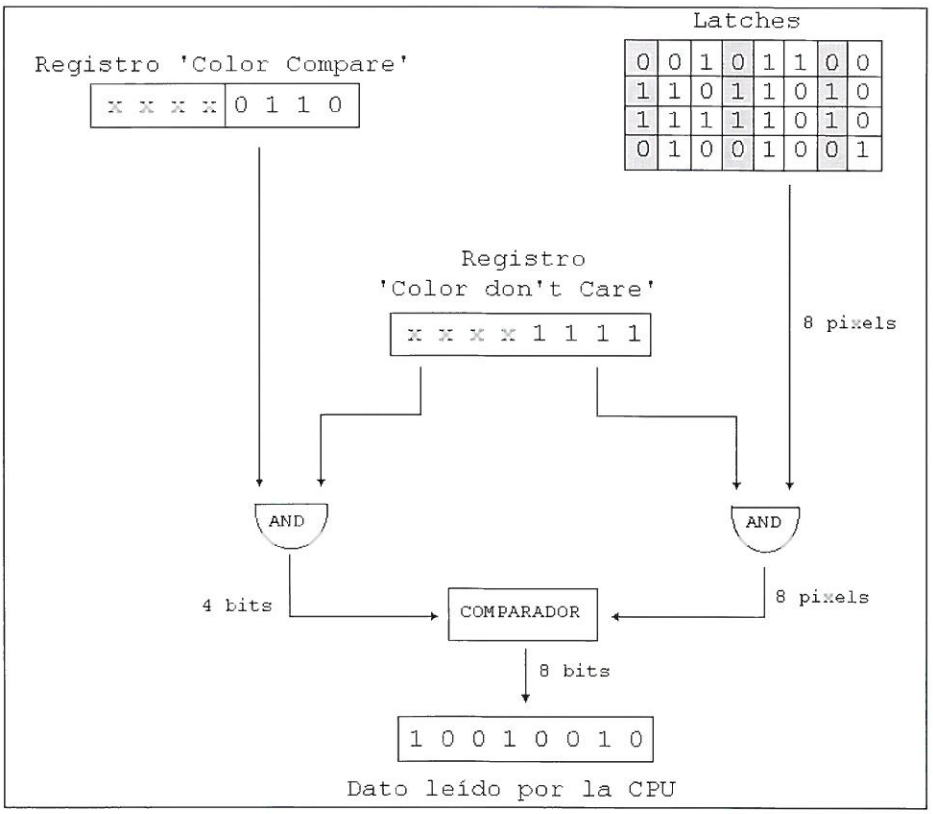

Latches
Registro 'Color Compare'
x x x x 0 1 1 0
0 0 1 0 1 1 0 0
1 1 0 1 1 0 1 0
1 1 1 1 1 0 1 0
0 1 0 0 1 0 0 1
Registro 'Color don't Care'
x x x x 1 1 1 1
8 pixels
AND
AND
4 bits
COMPARADOR
8 pixels
8 bits
1 0 0 1 0 0 1 0
Dato leído por la CPU

Figura 2. Modo 0 de Lectura.

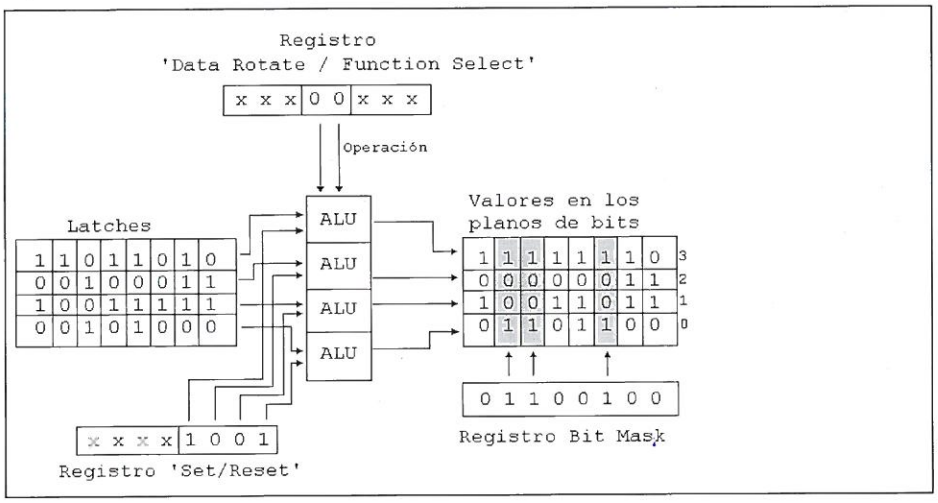


Fig.4 Modo 0 de Escritura. Valor del registro Enable Set/Reset = Fh

Rotate/Function Select.

El resultado de dicha operación para un determinado pixel se almacenará en los planos de bits siempre que en el registro *Bit Mask* el bit correspondiente dicho pixel esté activado (puesto a 1). Resulta obvio que estamos ante un modo orientado a pixels.

En la Tabla 2 se indican las distintas operaciones que se pueden realizar en las ALUs, y el correspondiente valor que se debe indicar en el registro *Data Rotate/Function Select.*

Cuando el registro *Enable Set/Reset* toma el valor 0, el comportamiento del modo 0 es diferente al hasta ahora descrito, tal como se puede apreciar en la Figura 5.

En este caso, las ALUs toman como operandos el valor ofrecido por la CPU y el contenido de cada uno de los *latches*. El resultado se almacena en los planos de bits, y, tal como sucedía antes, sólo se verán afectadas las posiciones en las que el registro *Bit Mask* valga 1.

El dato ofrecido por la CPU puede sufrir una modificación antes de pasar a ser operando de las ALUs. En efecto, se puede ver afectado por una rotación de sus bits si el valor de los bits 0-2 del registro *Data Rotate/Function Select* es distinto de 000b. El valor de estos bits indica el número de rotaciones que se realizan.

Observe el lector que este modo de operación está claramente orientado a bytes.

MODO 1 DE ESCRITURA

En este modo de escritura, el contenido de los *latches* (previamente cargados) se copia directamente en los planos de bits. En esta operación no interviene ningún registro del controlador de gráficos, ni el dato ofrecido por la CPU. (Ver Figura 6).

Este modo normalmente sólo se utiliza para transferencias de datos dentro de la memoria de video.

MODO 2 DE ESCRITURA

El modo 2 de escritura es muy similar al modo 0 de escritura cuando el registro *Enable Set/Reset* vale 0Fh. En este caso, el dato ofrecido por la CPU tiene las mismas funciones que el registro *Set/Reset* en el mencionado modo 0. Nótese que sólo se tienen en cuenta los cuatro bits menos significativos del byte de la CPU (Ver Figura 7).

En este modo no se utilizan los registros *Enable Set/Reset* y *Set/Reset*. Tampoco existe la posibilidad de rotar el valor suministrado por la CPU.

Una de las principales usos del modo 2 de escritura es la transferencia de imágenes almacenadas en la RAM del sistema en formato "empaquetado" hacia la memoria de video.

Aclaremos este punto: A veces, las imágenes que se guardan en archivos suelen tener un formato que, por sus características posee el calificativo de "empaquetado". En este formato, la imagen se almacena como una secuencia de pixels, en la que los 4 bits de cada pixel son contiguos.

Para transferir estas imágenes a la memoria de video y poder visualizarlas correctamente, es necesario tomar el valor de cada pixel y distribuir cada uno de sus bits al plano adecuado. Esto se consigue fácilmente usando el modo 2 de escritura y seleccionando la máscara (valor de *Bit Mask*) apropiada para cada pixel.

Es por esto que normalmente, y en aras de conseguir un mayor rendi-

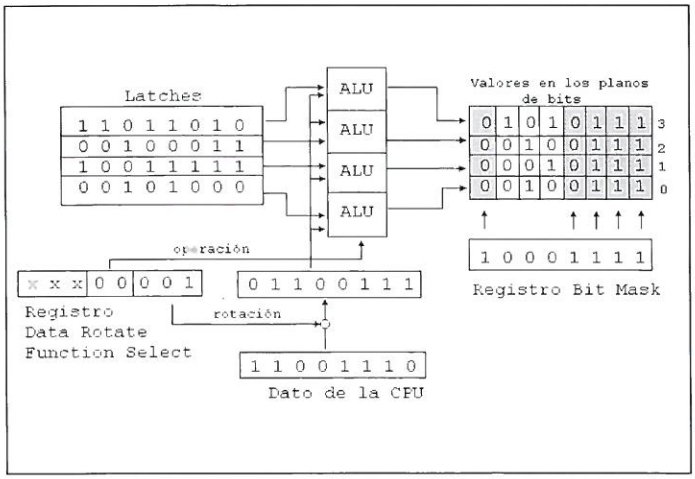


Fig. 5 Modo 0 de Escritura. Valor del registro Enable Set/Reset = 0h

El bus de datos interno de una VGA color es de 32 bits

El controlador gráfico dispone de 4 ALUs.

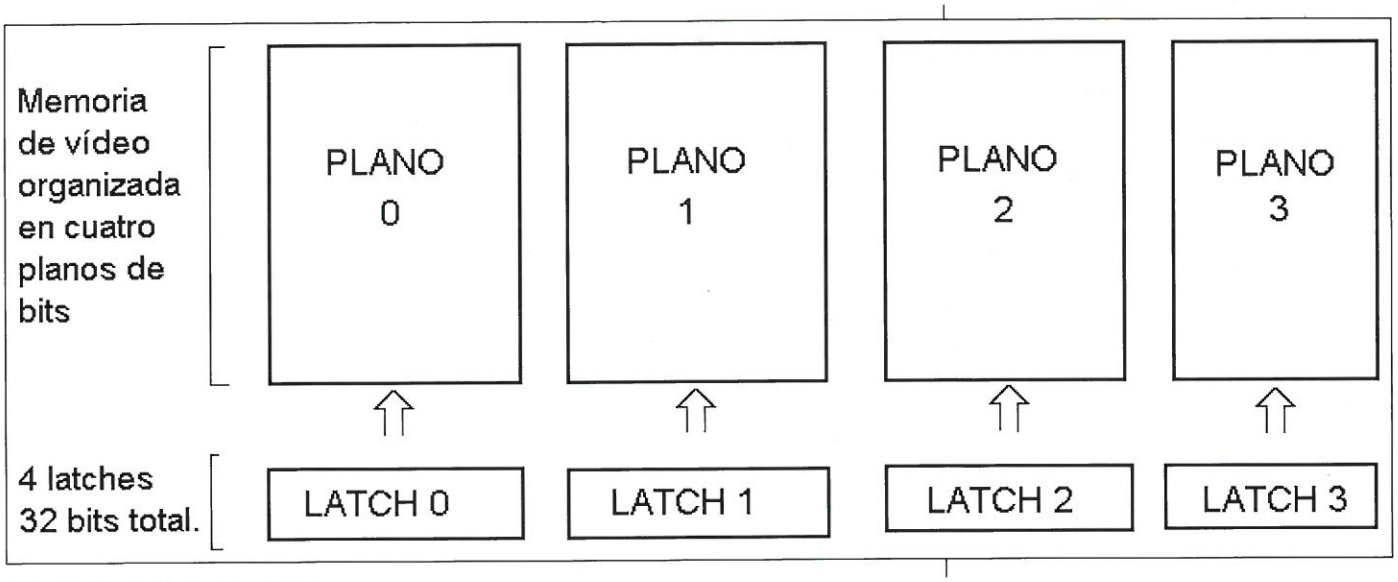


Fig. 6 Modo 1 de Escritura

miento, los bitmaps de 16 colores se suelen almacenar en disco como planos separados. Para restaurarlos en la memoria de video se suelen transferir línea a línea alternando los cuatro planos (para evitar los indeseables efectos transitorios de cambio de color que se podrían producir si volcamos directamente el contenido de cada uno de los planos).

MODO 3 DE ESCRITURA

El modo 3 de escritura es quizás el más complejo de los que se analizan aquí, puesto que en él intervienen prácticamente todos los elementos que hemos visto hasta el momento.

Conviene observar detenidamente la Figura 8, que ilustra todo el proceso de escritura en los planos de bits.

Como se puede comprobar, el

El controlador gráfico dispone de nueve registros

modo 3 es, en esencia, muy similar al modo 0 de escritura (cuando el valor del registro *Enable Set/Reset* toma el valor 0Fh). El registro *Set/Reset* y los bits 3 y 4 del registro *Data Rotate/Function Select* juegan exactamente el mismo papel que en aquel caso.

La novedad está en el uso que se hace del dato ofrecido por la CPU. En primer lugar, este dato está sometido a posibles rotaciones, según indiquen los bits 0-2 del registro *Data Rotate/Function Select*. El resultado de la rotación se utiliza como operando de una operación *AND* lógica con el valor del registro *Bit Mask*. Por último, el resultado de esta operación hace la función de máscara que en otros modos realiza directamente el registro *Bit Mask*.

Una de las principales ventajas de este modo es el hecho de poder variar dinámicamente la máscara que afecta a los cuatro planos de bits. De hecho, este modo se suele utilizar, entre otras cosas, para escribir múltiples pixels en la memoria de video, desde dentro de un bucle.

Otra ventaja, y quizás la más importante, no es muy evidente: el modo 3 suple la baja eficiencia de las operaciones de escritura en puertos (instrucciones *OUT*). En efecto, en el modo 0, cuando los datos a escribir varían mucho, será necesario realizar repetidas escrituras en los puertos para establecer los nuevos valores del registro *Set/Reset*. En esta situación es preferible que el dato lo ofrezca la CPU. Si

Existen 2 modos de lectura y 4 modos de escritura

¿SABE LO QUE SE PIERDE SI NO REGISTRA SUS APLICACIONES PARA MICROSOFT® WINDOWS 95?
✓ Soporte técnico gratuito por tiempo limitado.
✓ Suscripción gratuita a la revista de los Usuarios de productos Microsoft.
✓ Ofertas en actualizaciones, eventos, seminarios, cursos, etc.
✓ Tener la seguridad de que sus programas de software son legales.
Envíe ya su tarjeta de registro.
Para más información llámenos al telf.: (91) 804 00 96
Microsoft
¿HASTA DONDE QUIERES LLEGAR HOY?
PROG.

además es necesaria una rotación del dato, es asimismo deseable que dicha rotación la realice la CPU para evitar accesos a los puertos para modificar el valor del registro *Data Rotate/Function Select*.

Las características de este modo lo hacen especialmente indicado para el dibujo de líneas y para mostrar texto transparente (que no modifica el fondo sobre el que se muestra).

OTRAS CONSIDERACIONES

Es importante recalcar que en los modos nativos de la VGA no es posible leer directamente el valor de un pixel de la memoria de video.

Para conseguirlo, es necesario realizar una serie de operaciones: En primer lugar se debe activar el modo 0 de lectura, y a continuación se debe calcular la dirección de memoria en la que se encuentra el pixel buscado. Por último, desde dentro de un bucle que varíe el valor del registro *Read Map Select*, se deben leer secuencialmente los cuatro valores de los *latches*, para posteriormente aislar los bits que nos interesan y reconstruir el valor buscado.

A la vista del funcionamiento de los distintos modos, se observa que es posible mover datos dentro de la memoria de video de dos formas: utilizando directamente el modo 1 de escritura (cargando previamente en los *latches* los valores a mover), o bien usando el modo 0 de escritura teniendo cuidado de que el registro *Bit Mask* contenga el valor 0h (de esta forma no interviene el valor del registro *Set/Reset* en la operación).

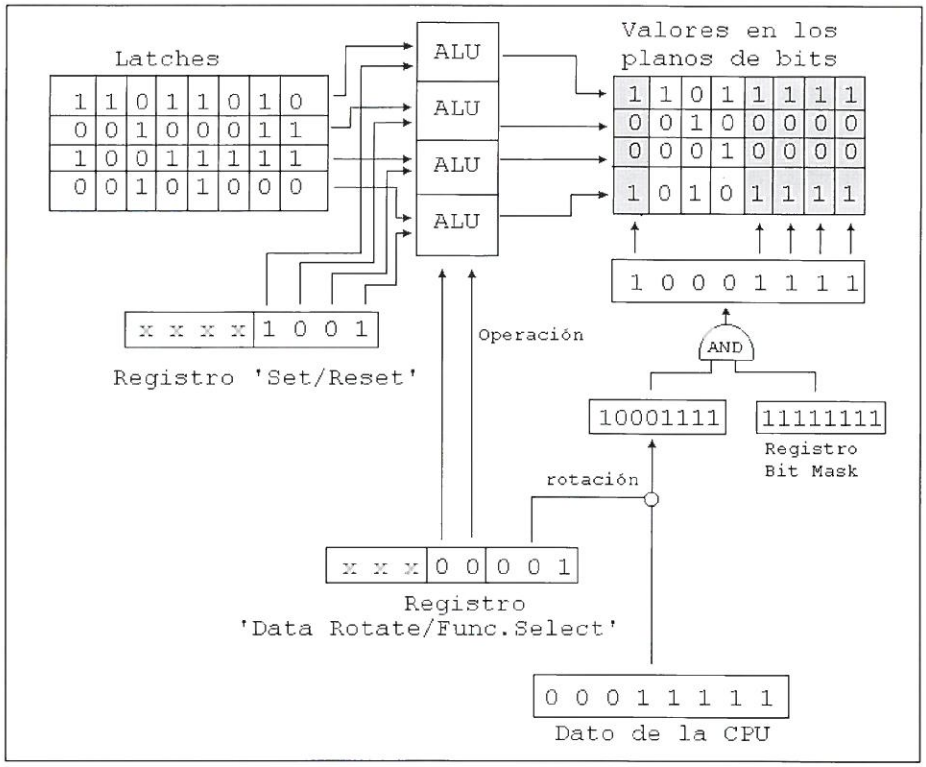

Latches
ALU
ALU
ALU
ALU
Valores en los planos de bits
1 1 0 1 1 0 1 0
0 0 1 0 0 0 1 1
1 0 0 1 1 1 1 1
0 0 1 0 1 0 0 0
1 1 0 1 1 1 1 1
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
1 0 1 0 1 1 1 1
1 0 0 0 1 1 1 1
x x x x 1 0 0 1
Registro 'Set/Reset'
Operación
AND
10001111
11111111
Registro Bit Mask
rotación
x x x 0 0 0 0 1
Registro 'Data Rotate/Func.Select'
0 0 0 1 1 1 1 1
Dato de la CPU

Fig. 8 Modo 3 de Escritura

de la BIOS para mostrar texto o pixels.

Es importante también ser conscientes de la potencia y flexibilidad que nos ofrece el registro *Bit Mask* (en sus dos variantes: manipulación directa o mediatizada por el dato de la CPU en el modo 3 de escritura). En efecto, la adecuada manipulación de este registro nos permite construir rutinas sumamente rápidas para dibujar puntos aislados, y, sobre todo, para dibujar líneas.

No es posible leer directamente el valor de un pixel

Se deberá tener especial cuidado a la hora de manipular los registros del controlador gráfico. Es necesario volver a poner sus valores por defecto antes de utilizar las rutinas

A la vista de todo lo aquí explicado se deduce que la gestión de los modos gráficos de 16 colores es bastante más compleja que la del modo 13h de 256 colores. En la próxima entrega de este curso seguiremos profundizando en el estudio de los modos gráficos de la VGA.

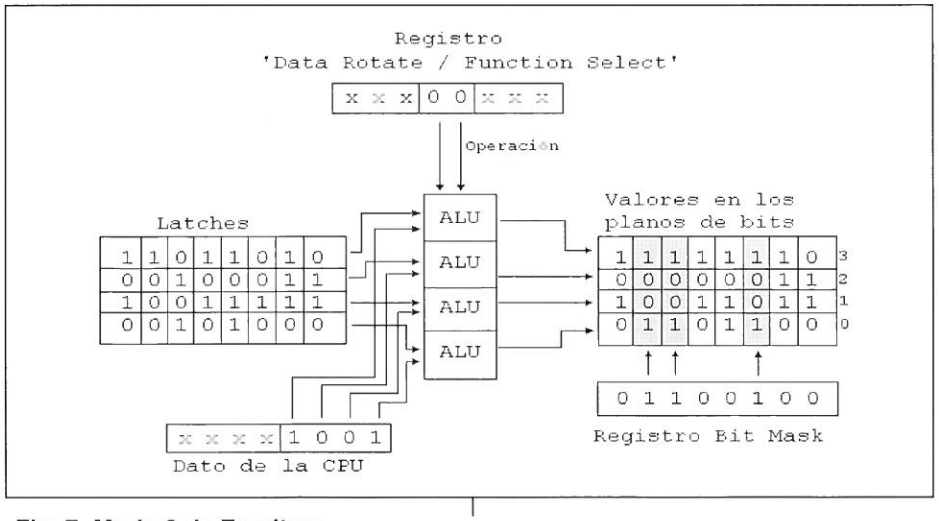

Registro
'Data Rotate / Function Select'
x x x 0 0 x x x
Operación
Latches
ALU
ALU
ALU
ALU
Valores en los planos de bits
1 1 0 1 1 0 1 0
0 0 1 0 0 0 1 1
1 0 0 1 1 1 1 1
0 0 1 0 1 0 0 0
1 1 1 1 1 1 1 0 3
0 0 0 0 0 0 1 1 2
1 0 0 1 1 0 1 1 1
0 1 1 0 1 1 0 0 0
0 1 1 0 0 1 0 0
Registro Bit Mask
x x x x 1 0 0 1
Dato de la CPU

Fig. 7 Modo 2 de Escritura

MEMORIA EXPANDIDA

Enrique de Alarcón

Prácticamente, todos los programas del mercado que aparecen desde hace años, usan funciones de algún gestor de memoria para superar la barrera de los 640 Kbytes de RAM disponible que da el DOS.

Uno de los gestores más estándar y veteranos, puesto que es el primero que se creo, es el *driver* de memoria *LIM-EMS* (*Expanded Memory Specification*) creado en los tiempos del 8086 por las empresas Lotus, Intel y Microsoft (*LIM*) y que va actualmente por la versión 4.0.

Si se es programador sin muchas necesidades y se va a crear una aplicación que va a funcionar bajo DOS en modo real 8086 y se decide usar el gestor de memoria *LIM-EMS* (cosa casi segura en estas condiciones), debe conocerse el modo interno de operar de este gestor, el modo en que controla la memoria y todos los servicios que pone a disposición del programador. En este artículo se explican todos los datos técnicos necesarios para que el programador pueda usarlo sin ningún problema.

ORÍGENES DEL GESTOR LIM-EMS

La causa de que se uniesen las tres empresas citadas para la creación de este estándar en gestión de memoria fue cuando Lotus, creadora de la famosa hoja de cálculo Lotus 1-2-3, comenzó a tener problemas para que sus usuarios pudieran crear hojas de cálculo grandes bajo su programa, que rápidamente consumía los escasos Kbytes de memoria que le ofrecía el DOS. Al mismo tiempo, empezaron las quejas de los programadores en general por las limitaciones que tenían a la hora de crear sus programas. Por eso, qué mejor para crear un estándar lo antes posible que unirse con el creador del sistema operativo sobre el cual se basaba (Microsoft) y el creador del hardware sobre el cual funcionaba su programa (Intel), con lo cual, todas las demás empresas y desarrolladores tendrían que aceptar el resultado.

Inicialmente, el gestor fue creado para soportar hasta un máximo de 8 Mbytes de memoria *EMS*, lo cual pareció más que suficiente en el momento de aparecer (igual que parecían suficientes los 256 Kbytes de RAM de los primeros ordenadores). Aunque con el paso del tiempo y el aumento de necesidades, esta cantidad se amplió hasta los 32 Mbytes de la versión 4.0.

Posteriormente, con la aparición de los 286 y al ampliar los buses de direcciones de los micros, se creó el *driver XMS* para el control de memoria física por encima del Megabyte, y que es el que se empezó a usar con la aparición de Windows. Pero esto ya es el tema del artículo del próximo número y su extensión es también considerable.

CARACTERÍSTICAS ESPECIALES

Al aparecer los primeros ordenadores con memoria *EMS* en tarjetas hardware, aparecieron diversas versiones del driver. Así por ejemplo, IBM tenía su propia forma de gestionar la memoria *EMS* integrando un controlador llamado *XMA*, y la casa *AST* tenía la llamada *EEMS*, una versión ampliada (*Enhaced*, mejorada) del gestor y los cuales, además de otros menos conocidos, no eran compatibles con el *LIM-EMS*, actualmente estándar indiscutible.

Además, también había tarjetas que usaban marcos de página de tamaños diferentes al estándar definido por Intel, habiendo así por ejemplo, ordenadores que permitían marcos de hasta 256 Kbytes o más (16 páginas físicas).

Pese a todos estos problemas iniciales, el programador no debe preocuparse, ya que estas versiones ya hace años

Gracias a la memoria *LIM-EMS*, las aplicaciones DOS en modo real pueden superar la limitación de los 640 Kbytes de memoria disponible bajo el mismo. Por tanto, es fácil deducir que, para poder crear aplicaciones mínimamente profesionales en DOS modo real, debe saberse usar este gestor de memoria creado por las empresas Lotus, Microsoft e Intel que ya es un estándar desde hace años.

que no se ven y es casi seguro que el lector no dispone de una de ellas.

PRIMER PASO

Antes de programar nada ni de realizar pruebas, debe tenerse instalado el gestor de memoria *EMS* en la memoria, el cual no es una librería a enlazar con nuestras aplicaciones durante el montaje de las mismas o un fichero binario a cargar con el código ejecutable a usar, si no que se carga como un controlador de dispositivo en el fichero de configuración *CONFIG.SYS* del DOS usando una línea del tipo *DEVICE=C:\DOS\EMM386.EXE* y el cual ya viene con todas las versiones del sistema operativo DOS desde la versión 4.0 del mismo.

Además, en la versión 6.0 y superiores del MS-DOS, se incluye el *MEMMAKER*, que ayudará a añadir parámetros al controlador automáticamente, optimizándolo para aprovechar mejor la memoria para funcionar bajo Windows y otras características, como aprovechar la zona de memoria de vídeo monocroma no usada normalmente en lo adaptadores en color.

CÓMO FUNCIONA LA MEMORIA EMS

Toda la memoria por encima del primer Mega que esté configurada como *EMS*, se divide en bloques de 16 Kbytes llamados *<páginas lógicas>*. Por lo tanto, si se tiene por ejemplo 1 Mbyte de memoria asignada como *EMS*, se dispone de 64 *páginas lógicas* de 16 kbytes cada una. La versión 4.00 del gestor *EMS*, permite un máximo de 32 Mbytes de memoria *EMS*.

Como se sabe, desde una aplicación en modo real, no se puede acceder a la memoria que está por encima del primer Megabyte, que es donde está la memoria *EMS*. Por esa razón, el gestor da un bloque de 64 Kbytes dentro del 1er Mbyte llamado *<marco de página>* donde hay capacidad para cuatro *<páginas físicas>* de 16 Kbytes que pueden usarse como *ventana virtual* (4 *páginas virtuales*), ya que a éstas si es posible acceder desde una aplicación y funcionarán como enlace entre la memoria de la aplicación y la memoria *EMS* disponible del gestor. Señalar que este marco de página, siempre se crea dentro del rango de memoria asignado a las extensiones de la BIOS (entre 768 Kbytes y 960 Kbytes dentro del 1er Mega) siempre y cuando se este hablando de un marco estándar de 64 Kbytes.

Cuando se quiera realizar alguna operación de lectura/escritura sobre una *página lógica*, debe copiarse primero a una de las *páginas físicas* (que funcionan como páginas temporales) mediante un servicio del gestor y realizar los cambios sobre ésta.

Los cambios realizados sobre la *página física*, se actualizan también sobre la *página lógica* a la cual corresponde (operación de la cual se encarga de realizar automáticamente el propio gestor de memoria).

Para referenciar a cualquier *página lógica* en las llamadas a los servicios del gestor, debe usarse el *handle* que proporciona el propio gestor al asignar la(s) página(s), al estilo de como deben usarse los *handles* de los ficheros abiertos en los servicios del DOS.

CÓMO USAR EL GESTOR

El gestor de memoria *EMS* funciona de forma parecida a otros controladores, como por ejemplo el del ratón, accediendo a sus servicios mediante una interrupción determinada y dando los parámetros mediante registros. Las versiones anteriores a la 4.00 disponían de 15 funciones y, a partir de la misma, se amplia el número hasta 30.

Todas las funciones se llaman mediante la interrupción 67h, y el número de servicio al que se quiere acceder se indica en AH (en AL se indica el número de subfunción si que es hay subfunciones). A si mismo, todas las funciones devuelven el resultado de la ejecución en el registro AH. Si AH=0, la ejecución de la función ha sido correcta, mientras que si AH=80h, se ha producido un error en el gestor, si AH=81h, se ha producido un error en el Hardware y si AH=84h, indica que se ha dado un número de función no válido, es decir, un valor sin signo menor de 40h o mayor de 5Dh o una función no soportada en la versión que se esté usando del gestor.

CÓMO INICIALIZAR EL GESTOR

Antes de empezar a usar las funciones del gestor *EMS* dentro de una aplicación, debe conocerse si ésta está o no presente, si tiene algún error de funcionamiento y donde está la dirección física del marco de página.

La memoria EMS se divide en bloques de 16 Kbytes llamados páginas lógicas

Para detectar si está o no cargado el gestor de memoria , hay que buscar la cadena ASCII *<EMMXXXX0>* a partir del *offset* 10 del inicio del código ejecutable del gestor. La dirección de inicio del *driver* en memoria, se encontrará en la tabla de vectores de interrupción (en el vector 67h). Esta tabla empieza en el segmento 0 y cada vector ocupa 4 bytes de definición (*SEGMENTO:OFFSET*), aunque puede buscarse también usando un servicio del DOS, que es mucho más cómodo (servicio 35h). Una vez que se sabe que está cargado (que se ha encontrado dicha cadena), hay que comprobar si posee algún error de funcionamiento, lo cual se hará llamando a la función 40h del gestor. Finalmente, si no se detecta error alguno, debe encontrarse la dirección inicial del marco de página llamando a la función 41h. En el código de ejemplo, si todo va bien y todo da correcto, devolverá en AX el valor 1. En caso contrario, devolverá AX=0. El código completo del procedimiento para detectar/inicializar el gestor quedaría como sigue:

```
INICIALIZA_EMS proc
; Detecta si gestor cargado en memoria
 mov ah,35h
 mov al,67h
 int 21h
    mov ax,es:[10]
    cmp ah,iMi
     jne DETC_EMS_NO
    cmp al,iEi
     jne DETC_EMS_NO
    mov ax,es:[12]
    cmp ah,iXi
```

```
    jne DETC_EMS_NO
    cmp al,iMi
    jne DETC_EMS_NO

; Detecta error de funcionamiento
mov ah,40h
 int 67h
cmp ah,0
jne DETC_EMS_NO

;Encuentra segmento marco página
mov ah,41h
int 67h
mov MARCO_PAGINA,bx
mov ax,1
ret

 DETC_EMS_NO:
  sub ax,ax
  ret
INICIALIZA_EMS endp
```

FUNCIONES DISPONIBLES

Pese a que el gestor proporciona 30 funciones, sólo se necesitan las primeras 6 funciones para la mayoría de aplicaciones que pueden crearse en un caso normal. Por eso, sólo se explicarán detalladamente estas primeras seis funciones. Además, las últimas funciones no se explican puesto que están diseñadas para ser usadas desde sistemas operativos y salen del alcance del presente artículo.

Todas las funciones hasta la 4Dh están disponibles en la versión 3.00 del gestor EMS:

Función 40h: Devuelve el estado del gestor de memoria *EMS*. Debe llamarse a esta función antes de usar ningún otro servicio y después de haber detectado su presencia. Si devuelve AH=0, todo está bien y puede utilizarse el gestor. Cualquier otro valor significa que se ha detectado algún error en su funcionamiento y por lo tanto, no está disponible para su uso. Código:

```
    INFORMA_ESTADO_EMS proc
    mov ah,40h
    int 67h
    ret
   INFORMA_ESTADO_EMS endp
```

Función 41h: Obtener la dirección inicial del marco de página (segmento dirección física al inicio de la primera página física). Salida: BX= Segmento (modo real) al inicio del marco de página (*ventana virtual*), considerando offset=0. El código queda como sigue:

```
   SITUA_MARCO_EMS proc
    mov ah,41h
    int 67h
    ret
   SITUA_MARCO_EMS endp
```

Función 42h: Obtener el número total de páginas lógicas y número de libres de que puede disponer la aplicación. Salida BX=Número de páginas libres de memoria *EMS*, DX=Número total de páginas. El código queda como sigue:

```
   INFORMA_PAGINAS_EMS proc
    mov ah,42h
    int 67h
    ret
   INFORMA_PAGINAS_EMS endp
```

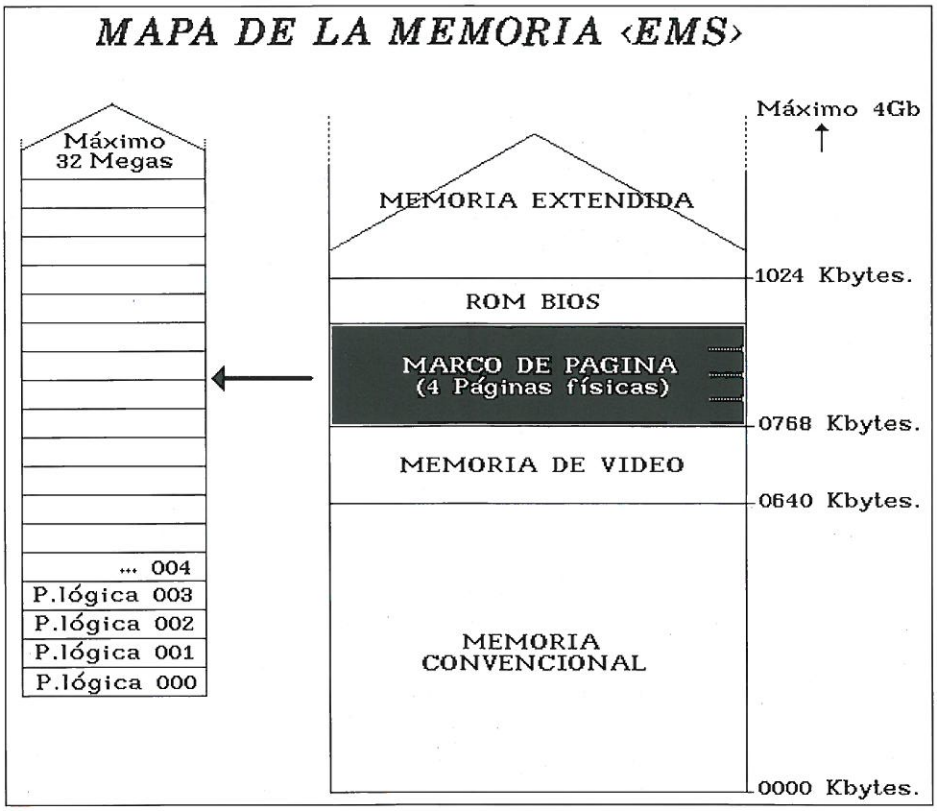
Función 43h: Asignar una o más páginas lógicas libres para nuestra aplicación. Entrada: BX=Número de páginas que se desean asignar. Salida: DX=*Handle* que corresponde a las página lógicas asignadas. El código queda como sigue:

```
   ASIGNA_PAGINAS_EMS proc
    mov ah,43h
    int 67h
    ret
   ASIGNA_PAGINAS_EMS endp
```

Función 44h: Copiar una página lógica en la página física indicada. Entrada: AL=Página física donde se copiará la página lógica , BX=Número de la página lógica que se va a copiar en la física. DX=*Handle* al bloque donde está incluida la página lógica indicada. Código:

```
   ACTIVA_PAGINA_EMS proc
    mov ah,44h
    int 67h
    ret
   ACTIVA_PAGINA_EMS endp
```

Función 45h: Libera páginas lógicas anteriormente asignadas por la aplicación. Entrada: DX=*Handle* correspondiente a las páginas que se desean copiar. El código es:



Esquema con todos los tipos de memoria.

```
LIBERA_PAGINAS_EMS proc
 mov ah,45h
 int 67h
 ret
LIBERA_PAGINAS_EMS endp
```

Función 46h: Devuelve la versión que se está usando del gestor de memoria *EMS*. Salida: AL=Número de versión en formato *BCD*. (Los 4 bits superiores corresponden a la parte entera de la versión y los 4 restantes son la parte decimal). Esta función es necesaria si se desea usar funciones disponibles sólo en la versión 4.00 y no se sabe si están disponibles. En caso de ser la versión 4.00 , devolverá el valor 40h.

```
VERSION_EMS proc
 mov ah,46h
 int 67h
 ret
VERSION_EMS endp
```

Las funciones restantes, no son necesarias excepto raras excepciones:

Función 47h: Salvar el mapa de las páginas lógicas indicadas.

Función 48h: Restaurar el mapa de las páginas lógicas indicadas.

Funciones 49h/4Ah: No usables por el programador, internas del gestor de memoria

Función 4Bh: Devuelve el número de *handles* activos (bloques de páginas lógicas asignados).

Función 4Ch: Devuelve el número de páginas lógicas que incluye el *Handle* que se le indique.

Función 4Dh: Devuelve el número de páginas lógicas asignadas a todos los *Handles* asignados por la aplicación.

Función 4Eh: Modificar el mapa de las páginas indicadas.

Funciones disponibles sólo a partir de la versión 4.0 del gestor EMS:

Función 4Fh: Definir un mapa parcial de las páginas indicadas.

Función 50h: Desactivar las páginas indicadas.

Función 51h: Modifica las páginas lógicas asociadas a un *Handle*.

Función 52h: Cambia atributos.

Función 53h: Obtener el nombre asociado a un *Handle*.

Función 54h: Obtiene la lista de *handles* activos.

Funciones 55h/56h: Cambiar mapas y saltos.

Función 57h: Desplazar un bloque de memoria.

Función 58h: Devuelve la lista de direcciones físicas que corresponden a las páginas lógicas libres.

Funciones 59h-5Dh: Usadas sólo en programación de sistemas operativos.

FUNCIONAMIENTO DE UN EJEMPLO

A continuación, se va a explicar un poco como funcionaría un programa de ejemplo que usase los servicios *LIM EMS*. Por ejemplo, podría plantearse un programa muy simple que necesitase de todas las funciones básicas *EMS*, y que tratase de cargar en memoria *EMS* un fichero de texto de 25 Kbytes para después liberarlo. Los pasos a realizar son los siguientes:

1.- Detectar si está el *Driver* de memoria *LIM-EMS* cargado en memoria (usando el código anteriormente detallado) buscando la cadena ASCII a partir de la dirección indicada por el vector de interrupción 67h, el cual se pude obtener mediante el servicio 35h del DOS indicando el AL el vector que se desea obtener.
2.- Detectar si hay algún error de funcionamiento del gestor (Función 40h).
3.- Localizar el segmento correspondiente al marco de página. (devolverá un segmento real 16 bits normal como si se tratase de un bloque de memoria de 64 Kbytes que se hubiese creado con un servicio del DOS)
4.- Asignar 2 páginas lógicas donde se almacenará el texto (comprobar si había 2 páginas libres asignables, - devolverá un *Handle*). Deben asignarse 2 páginas, ya que sólo se pueden usar páginas de 16 Kbytes y, los 25 Kbytes sólo pueden meterse dentro de 2 bloques de 16.
5.- Copiar las dos páginas lógicas en dos páginas físicas del marco de página, es decir, "mapear" las dos páginas mediante el servicio 44h.
6.- Cargar el texto sobre las dos páginas físicas como si fuese un bloque de memoria DOS normal.
7.- Liberar las dos páginas lógicas mediante el servicio 45h (Liberar *Handle*).
8-Salir al DOS normalmente.

ÚLTIMOS DATOS DE INTERÉS

Hay un dato muy importante a considerar antes de salir de las aplicaciones propias y es que, antes de volver al DOS, deben liberarse todos los *Handles* asignados anteriormente, puesto que sino, las páginas lógicas implicadas quedarán sin liberar y por lo tanto, no podrán ser usadas por aplicaciones posteriores, lo que no será de mucho agrado para el usuario de las diversas aplicaciones, ya que tendrá que reinicializar el sistema cada vez que finalice una sesión con el programa.

También debe tenerse en cuenta que, cuando se tienen por ejemplo las 3 páginas lógicas de un *handle* sobre 3 páginas físicas y se desea operar sobre estos 48 Kbytes dentro del marco de página como memoria normal, deben tenerse las páginas copiadas por orden, con el fin de que queden encadenadas correctamente, ya que sino, habrá pequeños bloques de 16K de memoria desordenados que llevarían al desastre los datos contenidos al realizar manipulaciones sobre los mismos. Imagínense, por ejemplo, un fichero gráfico GIF de 60 Kb (4 páginas) copiado en el marco en el orden 1,3,2,0 y que se intentase leer su cabecera para obtener tamaños, paleta y demás. Los resultados en pantalla serían obvios.

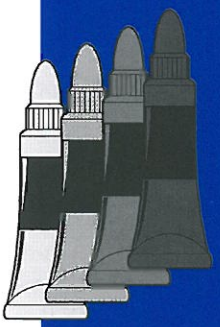
Otro detalle para programadores curiosos es que la memoria que crea el *EMM386.EXE* no es memoria *EMS* "real", sino que el gestor simula dicha memoria a partir de memoria *XMS* y esa es la causa de que deba estar cargado el controlador *HIMEM.SYS* para poder cargar el *EMMS386*.

RESUMEN

Primero se han explicado los orígenes históricos del propio gestor de memoria *LIM-EMS* y su evolución a lo largo del tiempo, así como los programadores que pueden necesitar de él. Después se ha detallado como se puede disponer de él cargándolo en el DOS desde el *CONFIG.SYS*. En la parte técnica, se ha explicado primeramente como gestiona la memoria el *LIM-EMS* así como la manera de llamar a sus servicios. Para terminar, se han detallado todos los servicios de que se dispone, así como ejemplos de código con las partes más importantes para dejar todos los detalles claros.

FORMATO PNG. EL GIF YA TIENE SUCESOR

Francisco Monteagudo Fernández



Hace ya tiempo que Compuserve, propietaria del formato gráfico GIF, anuncio su intención de cobrar derechos a aquellas firmas que incluyeran imágenes en este formato en sus programas. La comunidad informática respondió de forma previsible: crear un formato gráfico alternativo, el PNG

Compuserve creo el formato gráfico GIF en 1987 para resolver el problema de la portabilidad de imágenes entre ordenadores y de su envío a través de líneas telefónicas, problema que los formatos gráficos existentes no resolvían de forma satisfactoria. Así, algunos estaban ligados a determinadas arquitecturas hardware (el formato PCX, por ejemplo, está ligado al PC, mientras que el IFF está ligado al AMIGA), mientras que el formato portable por excelencia, el TIFF, era excesivamente complejo.

Hacía falta un formato que fuera simple pero eficaz, que se pudiera implementar sin dificultad para que se pudiera incluir en programas sharewa-re o de dominio público. Además, el formato debía estar preparado para el envío de imágenes a través de líneas telefónicas. La mayoría de los formatos no están preparados para ser leídos secuencialmente, es decir, si el fichero era enviado byte a byte a través de un canal secuencial (una línea telefónica), el receptor no podía irlo presentando en pantalla a medida que lo fuera recibiendo. Hechas todas estas consideraciones, se decidió que el GIF sería un formato para almacenar imágenes de hasta 256 colores, comprimidas con el algoritmo *LZW*, un algoritmo bastante eficaz pero sencillo de implementar. El resultado es que una rutina de decodificación de ficheros GIF podía implementarse en unos pocos centenares de líneas de código en C.

Compuserve todavía haría una segunda versión del formato en el año 89, añadiéndole capacidad para almacenar múltiples imágenes en un único fichero, aunque esta última característica apenas se ha utilizado.

El resto de la historia es conocido. El formato GIF se convirtió en el estándar para distribución de imágenes a través de BBS y en CD-ROM. Durante estos años, sólo ha habido un formato que le haya igualado en popularidad, el JPEG. Este último está concebido para almacenar imágenes a color real (característica no soportada por el GIF) comprimidas mediante el algoritmo de los cosenos discretos, algoritmo que permite unos niveles de compresión enormes a costa de una pérdida de calidad en la imagen. El JPEG es ideal para almacenar imágenes fotográficas, mientras que el GIF es ideal para almacenar dibujos. Esta diferencia en los campos de aplicación ha garantizado una coexistencia pacífica entre ambos formatos.

COMPUSERVE CAMBIA DE POLÍTICA

Una de las claves del éxito del formato GIF es que Compuserve autorizó su libre utilización, permitiendo que cualquiera pudiera distribuir imágenes en este formato. Sin embargo, el 29 de Diciembre de 1994 Compuserve publico unos nuevos términos de licencia, exigiendo el cobro de derechos a aquellos que editaran imágenes en este formato. Para agravar la situación, Unisys, propietaria del algoritmo *LZW*, decidió hacer valer sus derechos y cobrar de aquellos programas que trabajaran con imágenes GIF, por hacer uso de este algoritmo de compresión. Todo esto indignó a la comunidad informática, cuya respuesta no se hizo esperar. Así, a principios de 1995, se puso en mar-

cha el proyecto *PNG* (*Portable Network Graphics*) formado por un grupo de expertos en gráficos de primera fila, como Lee Daniel Crocker, quien ya había trabajado en el desarrollo de los formatos GIF y JPEG. El objetivo de este proyecto era crear un formato gráfico totalmente de dominio público, que no contuviera ningún componente registrado, a fin de que pudiera ser utilizado libremente por cualquier desarrollador sin temor a que alguna empresa pudiera exigirle el pago de derechos.

LA HISTORIA VUELVE A EMPEZAR

A los desarrolladores del PNG se les presentaron los mismos problemas y necesidades que a los desarrolladores del GIF, aunque contaban con una ventaja, la experiencia. Han transcurrido 8 años desde su aparición, lo que ha permitido conocer sus limitaciones. Ciertamente, el formato GIF tiene algunos defectos. El más conocido (y criticado) es su incapacidad para almacenar imágenes en color real, aunque hay otros más sutiles y que lo hacen bastante menos portable de lo que se supone debería ser. Un ejemplo son los textos. El formato GIF permite insertar textos dentro de un fichero, para poder introducir comentarios o aclaraciones sobre la imagen. El problema es que cada modelo de ordenador maneja la información textual de forma diferente, de manera que un texto escrito en un PC no se podrá leer correctamente en, por ejemplo, un Macintosh.

Otro problema importante era el algoritmo de compresión. Como ya se ha mencionado, el objetivo prioritario del PNG era que lo sucedido con el GIF no se repitiera, lo que significaba que en este nuevo formato no debía utilizarse ningún componente protegido por patente. Esto significaba que no podían utilizar el algoritmo de compresión (el algoritmo *LZW*) utilizado por el GIF.

Esta situación hizo que se descartara la primera propuesta, consistente en crear una versión modificada del formato GIF. En lugar de esto, se decidió aprovechar toda la experiencia adquirida para desarrollar algo enteramente nuevo, que corrigiera todos los defectos del GIF y que cubriera todas las necesidades previsibles.

FIRMA DE UN FICHERO PNG

Los primeros 8 bytes de un fichero PNG contienen los siguientes valores:

decimal: 137, 80, 78, 71, 13, 10, 26, 10
hexa: 89, 50, 4E, 47, 0D, 0A, 1A, 0A
ASCII: í, P, N, G, \r, \n, ^Z, \n

Figura 1.

ESTRUCTURA DE UN FICHERO PNG

El PNG es un formato estructurado en registros. Cada fichero está formado por una firma, seguida por una serie de registros que contienen la descripción de la imagen, que se almacena comprimida. El algoritmo elegido finalmente fue el *Deflate*. Este algoritmo es el utilizado por el conocido programa compresor PKZIP, y combina una excelente capacidad de compresión (muy superior al *LZW*) con el hecho de ser de dominio público, lo que garantiza su libertad de uso. Otra característica de este formato es la forma como se almacenan los valores numéricos; aquellos valores que ocupan más de un byte se guardan en formato *MSB*, es decir, con el byte mas significativo en primer lugar. Además, los valores numéricos que incluyen decimales se almacenan en formato de coma fija, es decir, se guarda como un valor de 4 bytes que contiene el valor multiplicado por 100.000. Así, el valor 0'5 se almacenará como 50.000.

LA FIRMA

El incluir una firma en la cabecera del fichero permite que los programas lectores puedan identificarlo al primer vistazo, es decir, el programa puede saber si el fichero es válido sólo con examinar los primeros bytes. Esto por sí solo es razón mas que suficiente para incluir una firma, pero sus creadores querían ir mas allá. El PNG es un formato para enviar imágenes a través de canales de comunicación. En principio, los canales deben transmitir los bits de datos directamente, sin realizar ninguna manipulación. Por desgracia, no siempre es así. Por ejemplo, en Internet hay muchos enlaces que suponen que la información que deben transmitir es texto y hacen transformaciones en lo

TIPOS DE REGISTRO ESTÁNDAR

NOMBRE	FUNCIÓN
IHDR	Cabecera de la imagen.
IEND	Marca de fin de imagen.
PLTE	Paleta de colores.
IDAT	Datos de la imagen
cHRM	Descripción del color
gAMA	Corrección gamma de la imagen
sBIT	Bits significativos
bKGD	Color de fondo
hIST	Histograma de la paleta
tRNS	Valores de transparencia de la paleta
pHYs	Dimensiones de los pixels
tIME	Fecha y hora de la última modificación
tEXt	Texto insertado en la imagen
zTXt	Texto comprimido

Figura 2.

que suponen que son los códigos de fin de línea. Así, se decidió que la firma de un fichero PNG (que puede verse en la figura 1) fuera un test para comprobar la línea. Las comprobaciones insertadas son dos: el primer byte permite detectar si el canal trabaja a 7 bits, y la secuencia 13,10 (marca de fin de línea en MS-DOS) permite detectar si el canal transforma la información que considera texto.

ESTRUCTURA DE LOS REGISTROS

Un registro, comienza con un campo de longitud, que indica el tamaño en bytes del campo de datos. Le sigue el campo de identificación, que contiene un código formado por 4 letras que indica el tipo de registro (figura 2). Después viene el campo de datos, que contiene la información almacenada en el registro, y por último viene el CRC (la suma de comprobación), que está aplicado a los bytes de todos los campos anteriores, excepto al campo longitud. El CRC se ha incorporado para reforzar el carácter de protocolo de transmisión de imágenes, para permitir la detección de los errores durante la transmisión.

El elegir una estructura basada en registros ofrece dos importantes ventajas. La primera es que permite que el formato pueda ser ampliado fácilmente por el sencillo procedimiento de ir definiendo nuevos tipos de registros. El programa lector decodificará únicamente aquellos registros que reconoce,

ignorando el resto. Si la información contenida en los registros ignorados no es crítica (imprescindible para que la imagen se pueda ver correctamente), la imagen se visualizará sin problemas.

La segunda ventaja es que permite que cualquier fabricante de software pueda añadir extensiones propias al formato para incluir capacidades necesarias para sus aplicaciones. Como en el caso anterior, los programas lectores pueden ignorar los registros específicos. Un ejemplo de este último caso es el conocido programa de creación de fractales *FractInt*, que tiene definido un registro propio para guardar la fórmula matemática que genera la imagen.

TIPOS DE REGISTRO DISPONIBLES

Como ya se ha mencionado, cada registro contiene un campo identificador formado por cuatro letras que indica el tipo de información que contiene. En la figura 2 se pueden ver los tipos de registros estándar. Como puede observarse, en los códigos de identificación hay algunas letras en mayúscula y otras en minúscula. Esta característica no es un capricho de los diseñadores, sino que sirve para indicarle a los programas como deben interpretar aquellos registros que no reconozcan.

Si la primera letra está en mayúscula, indica que este tipo de registro es crítico, es decir, es un registro cuya decodificación es obligatoria para poder mostrar la imagen correctamente. Esto también quiere decir que los programas lectores no están obligados a reconocer aquellos tipos que, aunque pertenezcan a la especificación, tengan esta letra en minúsculas.

Si la segunda letra está en mayúscula, indica que el tipo de registro pertenece a la definición estándar, es decir, los tipos específicos creados por fabricantes deberán tener esta letra en minúscula. Esto garantiza que los nuevos tipos que se puedan añadir a la especificación nunca coincidirán con los tipos privados que hayan creado los fabricantes.

La tercera letra no tiene ningún significado asignado, y se ha decidido que deberá estar siempre en mayúsculas.

La cuarta letra, cuando es mayúscula, indica que este registro contiene información que depende del contenido de la imagen, de manera que si se altera ésta, el contenido de este registro también deberá variar. Cuando un editor de imágenes altere un fichero que contenga tipos de registros con la cuarta letra de su identificador en mayúscula, en el fichero resultante se deben eliminar aquellos de estos registros que no sea capaz de interpretar.

REGISTRO IHDR

Este registro es el primero tras la firma, y marca el comienzo de la imagen. Este registro contiene los siguientes campos:

Ancho	4 bytes
Alto	4 bytes
Profundidad	1 byte
Tipo de color	1 byte
Compresión	1 byte
Tipo de filtro	1 byte
Entrelazado	1 byte

Ancho y Alto: Especifican la anchura y altura de la imagen en pixels.
Profundidad: indica el número de bits por pixel; los valores admitidos son 1,2, 4, 8 y 16.
Tipo de color: Máscara de bits que indica el tipo de imagen contenida en el fichero. El bit 0 indica que la imagen tiene paleta de colores; el bit 1 indica que la imagen es a color y el bit 2 indica que la imagen incluye un canal *alpha*. Los valores permitidos en este campo son: 0, 2, 3, 4 y 6.

Las combinaciones de valores que aparecen en los campos *Profundidad* y *Tipo de color* describen el tipo de imagen contenida en el fichero. En la figura 3 pueden verse todos los tipos permitidos según los valores definidos.
Compresión: Indica el método de compresión utilizado en la imagen. Actualmente, el único valor autorizado es el 0, que corresponde al algoritmo *deflate*.
Tipo de filtro: Identifica el método de preprocesado que se aplicará a la imagen antes de comprimirla. Actualmente, el único valor definido es el 0, que corresponde al filtrado adaptativo por líneas.
Entrelazado: Si vale 1, indica que la imagen está almacenada en modo entrelazado.

TIPOS DE IMÁGEN SOPORTADOS EN PNG

Tipo de color Profundidad Tipo de imagen
0 1, 2, 4, 8, 16 Imagen a escala de grises. Profundidad
indica el número de bits por pixel.

2 8, 16 Imagen a color real. Profundidad
indica el número de bits por canal;
así, si vale 8 cada pixel ocupará
18 bits (3 bytes).

3 1, 2, 4, 8 Imagen de paleta. Profundidad indica el número de bits por pixel.

4 8, 16 Imagen a escala de grises con canal *alpha*. Cada pixel va seguido por el valor de transparencia.

6 8, 16 Imagen a color real con canal *alpha*. Cada pixel es una serie RGB seguida por el valor de transparencia.

Figura 3.

REGISTRO IEND

Este registro marca el final de una imagen PNG. Este registro debe situarse al final del fichero, y no contiene ninguna información (la longitud de su campo de datos es 0). Este registro se utiliza como marca para indicarle al lector el final de una imagen PNG. Esta marca puede ser necesaria si la imagen se transmite por un canal de comunicación.

EL REGISTRO PLTE

El formato PNG, como ya se ha dicho, soporta imágenes basadas en paleta de hasta 256 colores. La paleta se almacena en este registro, como una tabla de hasta 256 elementos, siendo cada elemento un componente RGB.

MODO ENTRELAZADO

1 6 4 6 2 6 4 6
7 7 7 7 7 7 7 7
5 6 5 6 5 6 5 6
7 7 7 7 7 7 7 7
3 6 4 6 3 6 4 6
7 7 7 7 7 7 7 7
5 6 5 6 5 6 5 6
7 7 7 7 7 7 7 7

Figura 4. Indicación de qué pixels se cargan en cada paso.

EL REGISTRO IDAT

Este registro contiene datos de la imagen propiamente dicha. La imagen primeramente se comprime y la información resultante se va guardando en registros *IDAT*. No hay definido un tamaño máximo para éstos, de manera que, en teoría, toda la imagen podría meterse en uno solo, aunque en la práctica se recomienda trabajar con registros de tamaño medio (unos pocos Kbs), que el decodificador pueda cargar en memoria sin dificultad. La compresión, como ya se ha dicho, se realiza mediante el algoritmo *deflate*, el utilizado en el conocido programa de compresión PKZIP. Para ser mas precisos, la compresión se realiza con la librería *zlib*, un paquete de rutinas de compresión de datos de dominio público que implementan el algoritmo *deflate*. Los creadores del PNG decidieron utilizar esta librería para simplificar el trabajo de los desarrolladores, evitándoles el tener que escribir las rutinas de compresión/descompresión.

Un decodificador PNG mínimo esta obligado a reconocer los registros *IHDR*, *PLTE*, *IDAT* e *IEND*. Como el lector habrá observado, estos registros son los únicos que tienen su primera letra (registro crítico) en mayúscula.

EL FILTRADO DE LA IMAGEN

Los creadores del formato PNG se dieron cuenta de que el grado de compresión de una imagen puede mejorarse si esta es sometida a un tratamiento previo, un preprocesado, antes de ser compactada. Para ello definieron 5 tipos de filtros. Estos filtros son fórmulas matemáticas que se aplican sobre los bytes de la imagen. La manera de aplicarlos consiste en colocar, antes del comienzo de cada línea de pixels un byte que indica el tipo de filtro aplicado a esa línea. Los tipos de filtros son:

0: Ningún filtro. La imagen pasa al compresor tal cual, sin ninguna clase de tratamiento.

1: Resta. El valor de salida es la resta entre el pixel actual y el anterior.

2: Superior. El valor de salida es la resta entre el pixel actual y el que se encuentra inmediatamente encima (en la misma posición, pero en la línea superior)

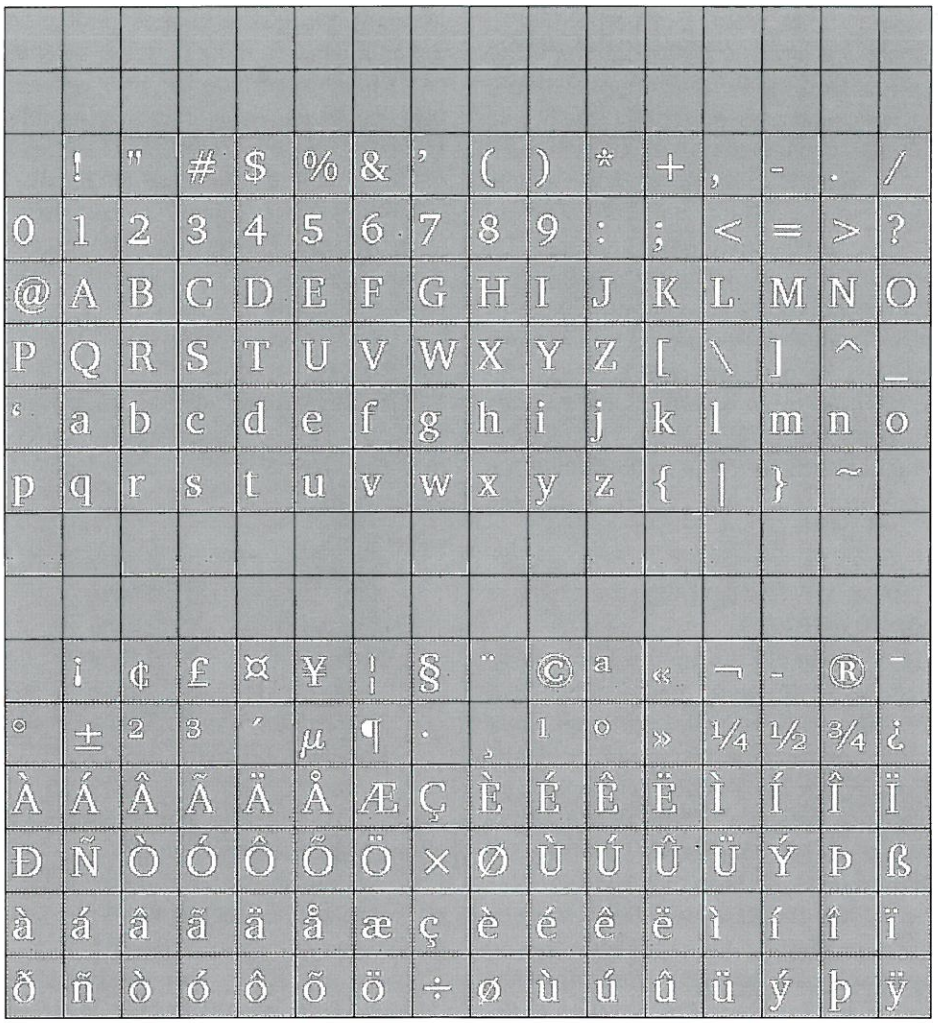


Figura 7: El juego de caracteres ISO8859-1.

3: Promedio. El valor de salida es el valor medio de dos pixels adyacentes.

4: *Paeth*. El valor de salida se obtiene tomando un cuadrado de 2x2 pixels en torno al pixel actual y aplicándole el algoritmo creado por Alan W. Paeth.

EL MODO ENTRELAZADO

El formato GIF tiene dos modos de almacenamiento de imágenes: el normal y en entrelazado. En modo normal, las imágenes se almacenan por líneas de pixels de forma consecutiva, es decir, primero se guarda la línea 0, luego la 1, etc. En modo entrelazado, las imágenes se dividen en cuatro bloques, conteniendo el primer bloque las líneas múltiplos de ocho, es decir, las líneas 0, 8, 16, 24, etc. Cuando se llega al final de la pantalla, comienza el segundo bloque que contiene las líneas 4, 12, 20, y así sucesivamente hasta completar la imagen. Este modo de guardar la imagen se introdujo para reforzar al GIF como protocolo de envío de imágenes. Enviar la imagen de esta forma permite que el receptor pueda ver en su pantalla como aparece rápidamente una primera imagen que, aunque es de muy poca calidad, permite hacerse una idea de su contenido. En los pasos sucesivos el receptor verá como la calidad de la imagen va mejorando hasta que se complete la transferencia. En el PNG se decidió reforzar esta capacidad mediante un entrelazado en dos dimensiones. La idea es suponer que la imagen está formada por cuadrados de 8x8 pixels cada uno, y la imagen se carga en 7 pasos, cargando en cada paso los pixels como se indica en la figura 4. Esto supone que si la imagen a cargar tiene una resolución de 640x480, es posible obtener de forma muy rápida una primera imagen de 80x60 pixels, que irá aumentando de resolución y calidad mientras se vaya cargando.

EL SOPORTE PARA CANALES ALPHA

El formato GIF permite definir un color como transparente. Esto quiere decir que aquellos pixels que tengan este color no serán pintados en pantalla, dejando lo que había dibujado antes. Esta característica permite que la imagen tenga contornos irregulares o huecos, lo que resulta muy útil en imágenes destinadas a insertarse en documentos o superponerse sobre otras imágenes. Los creadores del PNG decidieron mejorar esta capacidad añadiendo un canal *alpha*, que consiste en definir una máscara que contiene un byte por cada pixel de pantalla. Si el byte tiene el valor 0, significa que el pixel es transparente, es decir, deja ver lo que hay debajo. Si vale 255 significa que es opaco, es decir, se pintará con el color que tiene asignado. Por último, si tiene un valor intermedio, significa que es traslúcido, es decir, que el color del pixel se combinará con el color del fondo en la proporción especificada por este valor, dando un efecto de transparencia parcial o cristal coloreado. En las imágenes a color real o escala de grises, el canal *alpha* se almacena dentro de la misma imagen. Así, la longitud de los pixels se verá incrementada en un byte, para alojar el byte de la máscara. En cambio, en las imágenes de paleta de colores el canal *alpha* se almacena en el registro *tRNS*. Este registro tiene un byte por cada color de la paleta, conteniendo dicho byte el valor de transparencia para el color de la paleta que le corresponda.

LA RELACIÓN DE ASPECTO, EL REGISTRO pHYs

Los monitores de televisión tienen una relación altura/anchura de 3/4. Esto quiere decir que la altura de la imagen es igual a 3/4 de la anchura. Por desgracia, las resoluciones de las tarjetas de video no siempre respetan esta proporción. Por ejemplo, una resolución clásica en los PC es 320 pixels horizontales por 200 verticales. En esta resolución la proporción es de 2/3, lo que significa que los pixels son rectangulares, y que su altura será ligeramente mayor que su anchura. En cambio, otras resoluciones también muy utilizadas, como la de 640 horizontales por 480 verticales, sí que respetan la proporción de la pantalla, lo que significa que los pixels serán cuadrados. En estas condiciones, una imagen de 320x200 representada en una pantalla de 640x480 dará la sensación de estar aplastada. Para resolver este problema, el formato GIF incluye un indicador de la relación de aspecto, que indica la proporción vertical/horizontal que tiene la imagen, de modo que, si la pantalla sobre la que va a ser visualizada tiene una proporción distinta, esta pueda ser "reescalada" para no sufrir deformaciones. En el PNG esta característica se ha mejorado mediante la definición del registro *pHYs*, que indica las dimensiones del pixel, lo que permite conocer no sólo las proporciones de la imagen, sino también su tamaño exacto, lo que resulta de gran ayuda a la hora de imprimirlas sobre papel. El formato de este registro es:

4 bytes: Anchura del pixel
4 bytes: Altura del pixel
1 byte: Unidad de medida.

Si el campo *Unidad de Medida* vale 0, la unidad es desconocida, y las dimensiones sólo indican la relación de aspecto. Si vale 1, las dimensiones están expresadas en metros, en formato de coma fija.

TEXTOS EN LA IMAGEN. LOS REGISTROS tEXt Y zTXt

Estos registros permiten insertar textos en medio de la imagen. El texto a insertar se divide en dos partes. La primera parte es una palabra clave, que concluye con un código ASCII 0, seguida por el texto propiamente dicho. La diferencia entre ambos tipos de registros es que en el *zTXt* el texto principal, el que viene tras la palabra clave, se almacena comprimido, utilizando para ese fin, al igual que en la imagen, el algoritmo *deflate*.

El introducir una palabra clave permite identificar la información contenida en ese texto. Por ejemplo, para un programa que gestione una base de datos fotográfica resulta interesante que cada imagen contenga una ficha con sus datos técnicos. El esquema palabra *clave/texto* permite definir un registro de texto para cada campo de la ficha, siendo la palabra clave el nombre del campo y el texto el contenido de dicho campo. Esta solución tiene la ventaja de ser portable, ya que permite que cualquier aplicación pueda leer la ficha insertada en la imagen. En la figura 6 se puede ver la lista de palabras clave predefinidas, aunque cualquiera puede crearse las suyas propias.

Todavía queda un problema por resolver. Como se mencionó al principio de este artículo, cada modelo de ordenador tiene una forma diferente de manejar los textos, lo que hace que un fichero de texto creado, por ejemplo, en un PC no pueda ser leído correctamente en, por ejemplo, un Macintosh. Para resolver este problema, los creadores del PNG establecieron dos normas estándar para el manejo de los de los textos. La primera es que debe estar escrito usando el juego de caracteres ISO8859-1 (figura 7), el cual incluye todos los caracteres utilizados en los idiomas europeos, incluyendo la Ñ. Además de ser estándar en muchos sistemas operativos, como el UNIX o Microsoft Windows. La segunda norma es que en aquellos comentarios que contengan varias líneas de texto, la separación entre líneas se realizara mediante el código ASCII 10.

LA CORRECCIÓN GAMMA. EL REGISTRO gAMa.

Es sabido que en la pantalla de un ordenador todos los colores se obtienen mediante la combinación de los tres colores primarios, el rojo, el azul y el verde. De esta forma, cada color se describe mediante tres valores numéricos que indican los niveles de intensidad de cada color primario, correspondiendo el 0 a la ausencia de este color y el 255 al nivel máximo de brillo. Vista esta asignación de valores, en teoría el 128 debería corresponder a la mitad de intensidad, y de hecho es así en algunos sistemas (por ejemplo, el Macintosh), pero no en todos. En el PC, por ejemplo, el nivel de brillo aumenta siguiendo una curva exponencial. Para corregir este problema, se ha introducido la corrección *gamma*, que consiste en indicar el nivel de brillo correspondiente al nivel 128, y que sirve de referencia para cal-

cular los demás. El valor de la corrección *gamma* se guarda en un registro *gAMa* en formato de coma fija.

PALETAS, HISTOGRAMAS Y REDUCCIÓN DE COLORES. REGISTRO hIST

El formato PNG permite almacenar imágenes de paleta de hasta 256 colores; sin embargo, existen sistemas que no pueden presentar tantos colores en pantalla. Cuando el decodificador se encuentra con esta situación, debe llevar a cabo una reducción de colores, es decir, eliminar los colores menos utilizados y quedarse con los mas importantes. Para llevar a cabo este proceso, el primer paso es crear un histograma de colores, que es una tabla que contiene las frecuencias de aparición de los diferentes colores dentro de la imagen. Para facilitar el trabajo de los decodificadores, los creadores del PNG decidieron que este histograma se pudiera suministrar con la propia imagen, por lo que el registro *hIST*, que contiene, para cada color de la paleta, un valor de 16 bits que indica el grado de utilización de ese color dentro de la imagen.

EL REGISTRO sBIT

Como se puede observar, el formato PNG sólo soporta unas profundidades de color concretas. Así, para las imágenes de paleta sólo soporta 2, 4, 16 y 256 colores. Esto significa que una imagen de, por ejemplo, 32 colores (5 bits por pixel) se tendrá que guardar a 256 colores, es decir, que de los 8 bits que ocupa cada pixel, sólo 5 serán significativos. En una situación como esta, el creador de la imagen puede insertar un registro *sBIT* que contiene un valor de un byte si la imagen es de paleta o escala de grises, y tres bytes si es a color real que indica el número de bits significativos para los pixels.

EL REGISTRO bKGD

Al igual que en el formato GIF, en el PNG también es posible especificar un color de fondo, con el que se colorearán aquellas partes de la imagen que sean transparentes o aquellas áreas de la pantalla que la imagen no cubra. El valor de ese color se almacena en un registro de este tipo. Si la imagen es de paleta de color, este registro guarda un byte, y si es a color real, guarda un valor RGB.

EL REGISTRO tIME

Este registro contiene la fecha y hora de la última modificación de la imagen. Su formato es:

2 bytes: Año.
1 byte: mes (1-12).
1 byte: día (1-31).
1 byte: hora (0-23).
1 byte: mins.(0-59).
1 byte: segs.(0-60).

CARACTERÍSTICAS NO INCLUIDAS

El formato PNG, como sucesor del GIF, debería incluir todas las capacidades de este último. De todas maneras, como ya se mencionó, los creadores del PNG contaban con la ventaja de la experiencia, lo que les permitió, entre otras cosas, comprobar que algunas características incluidas en el GIF resultan innecesarias, por lo que se decidió prescindir de ellas. La primera es el número de versión. Los ficheros GIF llevan en su cabecera un número que indica con que versión del formato ha sido realizada. El programa lector comprueba este número, rechazando la imagen si no es capaz de interpretarla (si la versión de la imagen es posterior a la máxima que admite el programa). El problema de esta estrategia es que no permite compatibilidad descendente, es decir, una imagen que tenga como número de versión el 89, jamas podrá ser leída con un programa que sólo trabaje con imágenes versión 87.

Otra característica no incluida es la capacidad que tiene el GIF de almacenar varias imágenes en un solo fichero. Esta no se incluyó porque, aparte de que apenas se utiliza, complica considerablemente la creación de decodificadores. Los creadores del PNG han afirmado que quizá diseñen una versión especial del formato adaptada para almacenar múltiples imágenes por fichero, pero se trataría de un formato nuevo, es decir, el PNG es y será un formato monoimagen.

Tampoco se incluyó un algoritmo de compresión con pérdida. Este tipo de algoritmos permiten unos grados de compresión muy altos, aunque a costa de perder calidad en la imagen. El no incorporar un algoritmo de este tipo obedece a dos motivos. El primero es que complicaría extraordinariamente los decodificadores. El segundo es que, para el almacenamiento con perdida, ya existe el formato JPEG. Al menos por el momento, el *deflate* será el único algoritmo de compresión soportado por el PNG.

Lista de palabras clave predefinidas

PALABRA CLAVE	SIGNIFICADO
Title	Título de la imagen.
Author	Nombre del autor de la imagen.
Copyright	Mensaje de copyright.
Description	Descripción de la imagen.
Software	Programa usado para crear la imagen.
Disclaimer	Aclaraciones de tipo legal.
Warning	Advertencia sobre el contenido de la imagen.
Source	Dispositivo usado para generar la imagen.
Comment	Comentario de tipo general.

Figura 6.

PARA FINALIZAR

Los creadores del formato PNG no sólo han creado una alternativa al GIF, sino que han sabido corregir todas sus limitaciones y añadirle todo lo que le faltaba. El resultado es un formato con muchísimas mas prestaciones, aunque de una complejidad mayor. De todas maneras, aquellos programadores que deseen incluir imágenes PNG en sus aplicaciones no deben asustarse. Al estar basado en una librería estándar (la *zlib*) simplifica considerablemente el trabajo, eso sin contar con la estructura en registros de este formato, que permite prescindir de aquello que se sepa que no se va a utilizar.

Para contactar con el autor:
FidoNet: 2:343/140.22
Internet: monti@bbs-ce.uab.es

PROGRAMACIÓN MEDIANTE MFC

Jorge del Río

Desde la aparición de la librería de clases *MFC* para programación en Windows ha ido progresivamente evolucionando hacia un entorno de desarrollo sencillo y potente, donde la reutilización de código y la programación visual apartan progresiva y rápidamente la tortuosa programación en C del API

Un día en la vida de un programador se plantea, ya sea por necesidad o bien por "hobby", el hecho de realizar programas para Windows, y descubre todo un mundo de programación radicalmente diferente, basado en el tratamiento de eventos y no en la programación secuencial característica de MS-DOS. Además, según se avanza en el conocimiento del API de Windows y tras tortuosas sesiones detrás de la pantalla sin resultados aparentes, se progresa en la elaboración de programas con una apariencia realmente buena, sin invertir demasiado tiempo en la creación del interfáz gráfico, tal y como ocurría en DOS.

Después de realizar muchos programas más o menos complicados, se ve que siempre existe una parte común a todos ellos y que, de algún modo, debería automatizarse para no tener que repetir lo que sin duda llega a ser un poco pesado, el esqueleto de una aplicación Windows, es decir, la implementación de la función *WinMain*, el registro y la creación de la ventana principal, así como la implementación de la función de gestión de mensajes de dicha ventana principal. Un claro ejemplo de esqueleto de aplicación puede ser el pequeño programa del articulo anterior del curso.

Una solución no muy elegante, aunque si eficaz, es tener una especie de patrón desde donde se inician todas las aplicaciones, y que será copiada en cada proyecto. Adaptando los nombres, se consigue en muy poco tiempo la funcionalidad mínima de una aplicación. Pero un día aparece la librería *MFC* y se descubren las posibilidades brindadas mediante *AppWizard*, un verdadero mago para crear el esqueleto de las aplicaciones y que ha ido evolucionando hasta ser una herramienta casi increíble tanto por potencia como por versatilidad en la versión actual de Visual C++.

La librería *MFC* y las herramientas de Visual C++ 4.0 de automatización de programación basan todo su funcionamiento en las características aportadas por los lenguajes orientados a objetos. Por tanto, antes de aventurarse ha desarrollar con la *MFC*, deben tenerse unos conocimientos mínimos de programación orientada a objetos y su implementación en C++. El objetivo de este curso es conocer el compilador Visual C++ 4.0 y aplicarlo en la creación de aplicaciones para Windows, por lo que no se entrará a explicar nociones de programación orientada a objetos. De cualquier forma, es muy recomendable que el lector no iniciado en el tema se informe sobre el mismo para poder seguir el desarrollo del curso con éxito.

PROGRAMACIÓN EN WINDOWS UTILIZANDO LA LIBRERÍA MFC

Cuando se pasa de la programación para DOS a la programación para Windows mediante el API, debe enfocarse la metodología de creación de aplicaciones desde un punto de vista diametralmente opuesto. De algún modo, también existe un cambio radical al pasar a utilizar la *MFC* y *AppWizard*, ya que gran parte del código inicial en C++ de las aplicaciones es creado automáticamente por el propio compilador que, de algún modo, se traduce en un sentimiento de perdida total del control del desarrollo de la aplicación. Este sentimiento inicial, a medida que se

conoce el entorno de desarrollo, se diluye y se transforma en un sentimiento de satisfacción al comprobar la cantidad de trabajo ahorrado.

La librería de clases *MFC* encapsula gran parte del API de Windows, realizando gran parte de las operaciones repetitivas y ofreciendo una interfáz mucho más sencilla y rápida de utilizar, lo que se traduce en un incremento en el rendimiento en el desarrollo del programa. Existen un sin fin de clases para ser utilizadas, como por ejemplo la clase *CWinApp* (para la creación de la aplicación), la clase *CWnd* (para crear ventanas), clases para manejar los controles estándar de Windows y Windows'95, clases para manejar los objetos del GDI, para manejar objetos OLE, para acceder a bases de datos,...

Para comenzar a realizar una aplicación Windows con la *MFC*, se debe utilizar la herramienta *AppWizard*, que va a permitir elegir a la carta el modelo del esqueleto para nuestra aplicación.

CREACIÓN DE UNA APLICACIÓN CON APPWIZARD

El empleo del *AppWizard* ("Mago de Aplicaciones") es realmente práctico, ya que mediante esta herramienta puede elegirse un tipo de aplicación entre diferentes opciones, de la forma más sencilla e intuitiva posible, obteniéndose como resultado varios ficheros que serán el esqueleto de la aplicación.

Cuando *AppWizard* crea una aplicación, básicamente se generan los siguientes ficheros:

- Un fichero que contiene las clases que definen la aplicación y como iniciarla,
- otro que contiene las clases para gestionar los documentos,
- otro para gestionar la forma en la cual se muestran los contenidos de los documentos anteriores y
- un último fichero donde se encuentran las clases encargadas de crear la ventana principal con la cual interactuará el usuario.

Además, se generan los recursos estándar, incluyendo menús, la barra de herramientas, la barra de estado, etc. Por otra parte, también se pueden crear los ficheros *RTF* que servirán para crear el sistema de ayudas de la aplicación.

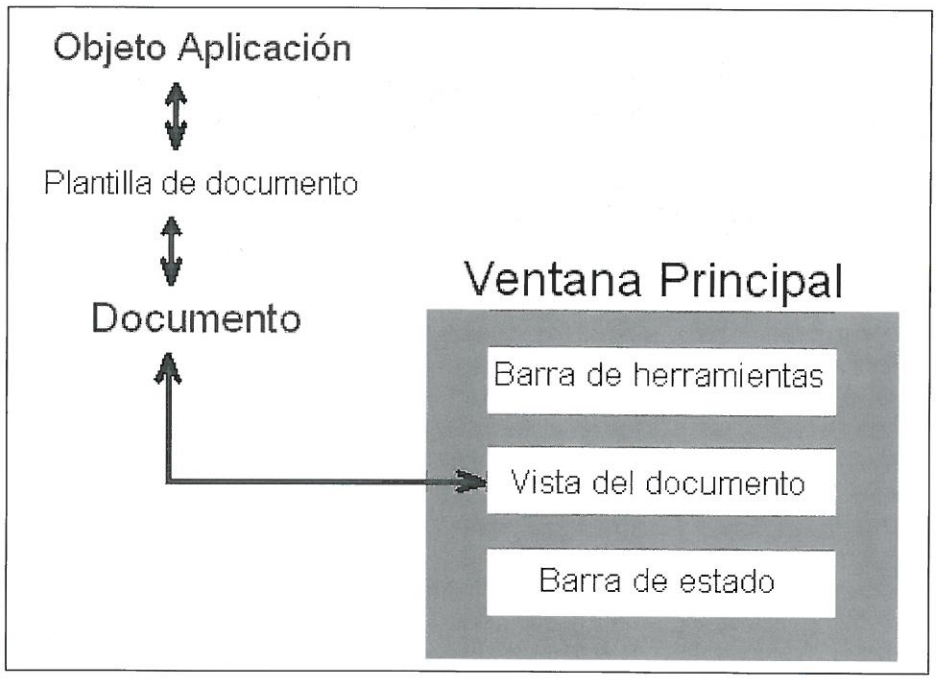


Figura A. Relación entre los objetos de una aplicación SDI.

ANTES DE GENERAR EL ESQUELETO DE LA APLICACIÓN

Antes de comenzar a entrar en el proceso de generación de una aplicación, deben fijarse una serie de conceptos sobre los cuales trabaja *AppWizard* a la hora de crear las aplicaciones.

Antes se ha hablado de los ficheros generados por *AppWizard*, entre los que se ha incluido uno que contiene las clases para gestionar los documentos. Dicha clase en la aplicación estará derivada de la clase *CDocument*, la cual se encarga de gestionar los datos de la aplicación, tanto para ser guardados como para ser leídos desde disco. Además, se hizo referencia a las clases para visualizar y presentar dichos datos al usuario, donde se utilizarán clases derivadas de *CView* o bien derivadas de algunas de las clases derivadas de la propia clase *CView*. Por ejemplo, si se utiliza una clase derivada de *CFormView*, podrá crearse una aplica-

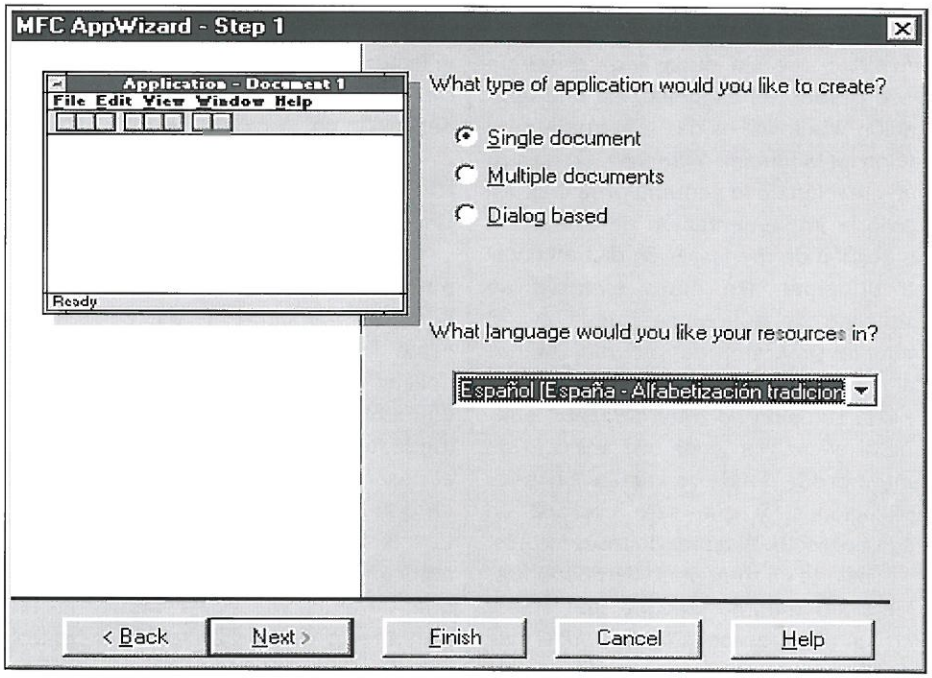


Figura B. Tipos de aplicación desde AppWizard.

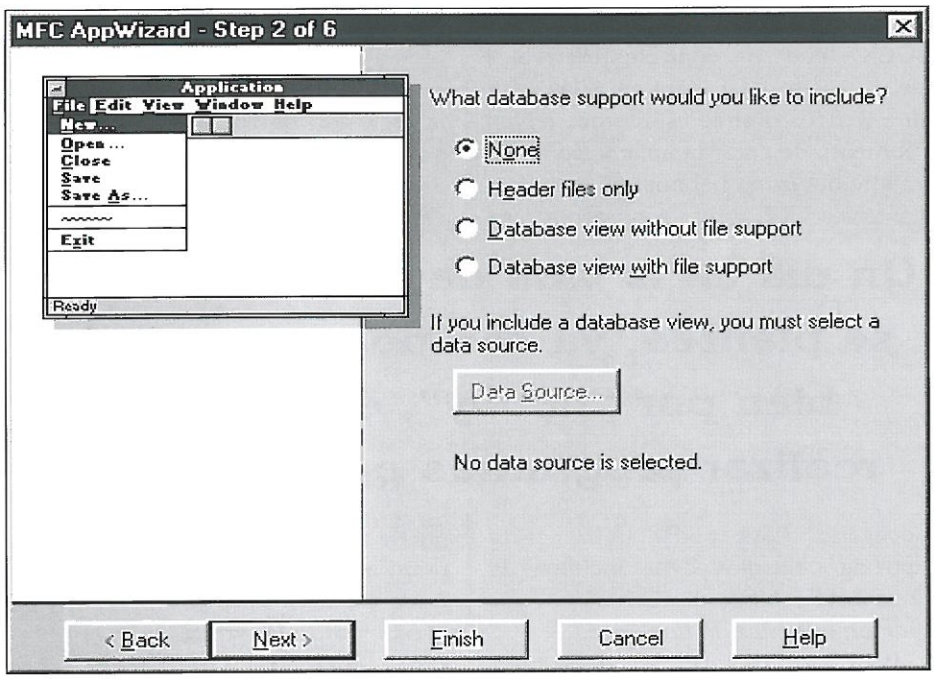


Figura C. Opciones para bases de datos.

ción similar a las creadas desde Visual Basic, basadas en un formulario donde se incluyen los diversos controles a utilizar. Si se utiliza una clase derivada de *CEditView* se creará un sencillo, pero con completa funcionalidad, editor de textos. Estas vistas deben estar incluidas dentro de la ventana o ventanas que forman el esqueleto, o *framework*. La aplicación se puede orientar de dos modos: como *SDI* (*Simple Document Interface*), donde sólo existe una ventana con la visualización del documento al mismo tiempo, o como *MDI* (*Multiple Document Interface*), donde coexisten dentro de la ventana principal todas las ventanas hijas, cada una con un documento diferente. Además, las llamadas plantillas de documentos, o *document templates*, permiten dirigir el proceso de creación de los documentos, vistas y el esqueleto de ventanas. Cada uno de estos elementos está derivado de la clase *CDocTemplate* y gestiona todos los tipos de documentos abiertos en la aplicación de este tipo. Por último, queda hablar de la clase de aplicación, derivada de *CWinApp* que sólo debe haber una por cada aplicación. Esta clase debe gestionar los procesos de inicialización y limpieza de todos los objetos de la aplicación de las clases anteriores, así como crear todos las plantillas de documentos que utilice la aplicación.

CREACIÓN DEL ESQUELETO DE NUESTRA APLICACIÓN MFC

En el siguiente apartado se va a tratar el método para crear una aplicación *MFC* desde *AppWizard*. Para ello, se va a crear un editor de textos, con capacidades totales de edición, impresión, previsualización de impresión y acceso a ficheros.

Al crear un nuevo proyecto, aparece una ventana para indicar qué tipo de proyecto se desea iniciar. En ella debe pulsarse sobre la opción *MFC Application(exe)* e indicar el nombre del proyecto. Posteriormente, se visualizarán las siguientes ventanas encargadas de mostrar todas las opciones posibles para crear el esqueleto de la aplicación.

Paso 1. Elección del tipo de aplicación.

En la primera ventana de *AppWizard*, se muestran tres opciones para elegir el tipo de la aplicación, ya sea *SDI*, *MDI*, o una aplicación basada en una ventana de diálogo. Además, puede elegirse el idioma en el cual se crearán todos los nombres de los menús, ayudas, etc., con la posibilidad de elegir el castellano. En la Figura B se muestra la elección hecha para el ejemplo, donde se ha elegido una ventana *SDI* con mensajes en castellano.

Paso 2. Soporte de bases de datos.

En esta segunda ventana se puede elegir el tipo de soporte para acceso a bases de datos de la aplicación. Las opciones ofrecidas son:

- Ningún tipo de acceso a bases de datos,
- inclusión sólo del fichero de cabecera *AFXDB.H* permitiendo implementar los métodos de acceso a los registros,

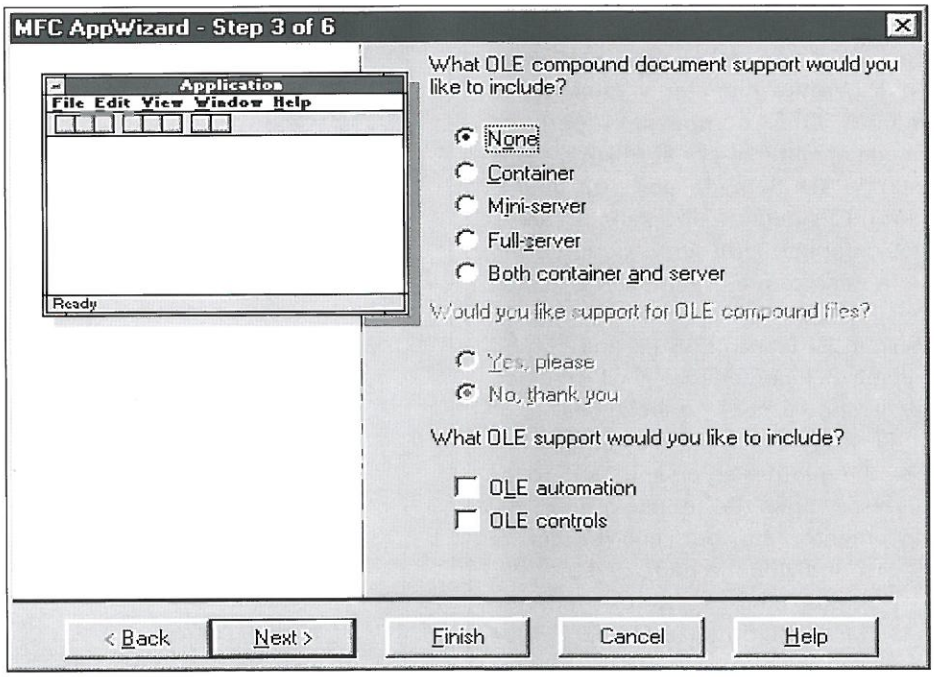


Figura D. Soporte de aplicaciones OLE.

- acceder a bases de datos sin acceso a ficheros
- con acceso a ficheros.

En estos dos últimas opciones se crea una aplicación con una vista del tipo *FORM* basada en la clase *CRecordView* y, también se crea una clase derivada de *CRecordset* para acceder a los registros de la base de datos. Además, en estas opciones debe indicarse el camino de la base de datos, así como el modo de enlace, ya sea por *ODBC* o mediante *DAO*. La Figura C ilustra la ventana con las opciones anteriores, donde se ha elegido "sin acceso a bases de datos".

Paso 3. Soporte OLE.

El soporte OLE está totalmente integrado, permitiendo acceder a toda la potencia de esta tecnología sin necesidad de un conocimiento absoluto, ya que es el propio Visual C++ el que lo realiza. Las tres partes principales son: el tipo de aplicación OLE, el soporte de ficheros compuestos (*compound files*) y soporte de automatización OLE y controles OLE, como se puede ver en la Figura D.

Mediante el tipo de aplicación OLE puede elegirse entre cuatro tipos de aplicación: una aplicación sin capacidades OLE, una aplicación del tipo contenedora (lo que implica poder añadir objetos OLE de otras aplicaciones en los documentos de la aplicación) o una aplicación *mini-server* (que permite manejar y crear documentos OLE compuestos pero no puede ejecutarse por sí misma, necesitando ser llamada por otra aplicación). El siguiente tipo sería un servidor completo (*full-server*) que permite lo mismo que el *mini-server* con la posibilidad de ejecutarse por si misma. El último tipo es una combinación del contenedor y el servidor, incluyendo ambas capacidades.

El soporte a ficheros compuestos permite guardar en un solo fichero los diversos tipos de objetos OLE del documento, sin por ello perder el acceso individual a cada objeto.

Los dos últimos puntos permiten habilitar la automatización OLE, que proporciona acceso a otros clientes de automatización a la aplicación y habilitar el manejo de controles OLE (OCX), que son controles utilizables de un modo casi idéntico a las VBX de *Win16*. Puede ser que en el momento de crear la aplicación no se estime oportuno habilitar el soporte a controles OLE y que a posteriori sea necesario. Para poder utilizar este tipo de controles debe incluirse la línea *AfxEnableControlContainer* en la función miembro *InitInstance*.

Para esta primera aplicación tampoco se ha incluido el soporte de aplicaciones OLE, las cuales serán tratadas con mayor detalle en números posteriores.

Paso 4. Capacidades de la aplicación.

En esta ventana (ver Figura E) se pueden elegir algunas de las capacidades de la ventana principal y la aplicación. La primera decisión a tomar es si se desea incluir barra de herramientas, que contiene los diferentes botones que suelen ir debajo del menú. El siguiente paso es decidir la inclusión de barra de estado, que permite visualizar de forma breve el estado de la aplicación. A continuación, es posible elegir si la aplicación puede o no imprimir y visualizar a priori la impresión los documentos. Además, es posible incluir soporte de ayuda, por lo que también será necesario crear los correspondientes ficheros. Otro aspecto a tener en cuenta es el soporte de los denominados controles en 3D, que ofrecen a las cajas de diálogo de Windows NT una apariencia tridimensional similar a la implementada en Windows'95.

Un día en la vida de un programador se plantea, ya sea por necesidad o bien por "hobby", el hecho de realizar programas para Windows

Hay dos componentes muy importantes que se pueden soportar, como son *MAPI* y los *Windows Sockects*, que permiten respectivamente a la aplicación gestionar los procesos de envío y recepción de correo electróni-

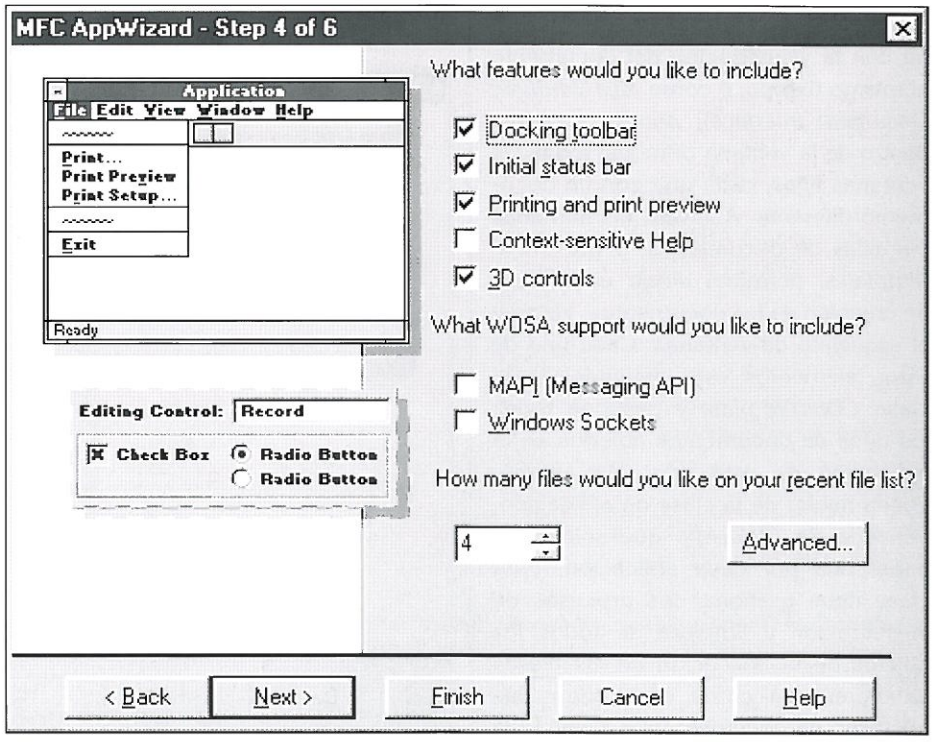


Figura E. Capacidades definibles en AppWizard de la aplicación.

co y las comunicaciones con otras maquinas mediante el protocolo TCP/IP.

En el apartado de avanzados, se puede nombrar a la aplicación, a los tipos de ficheros, fijar los filtros de las extensiones de los documentos, así como definir los estilos de la ventana principal de la aplicación.

Paso 5. Tipo de enlace de la librería *MFC* y comentarios.

En esta ventana se puede elegir la forma de enlazar las librería de clases con el código de la aplicación, ya sea de forma estática o bien como una librería dinámica. Esto último implica añadir determinadas DLLs al realizar copias de la aplicación. Por otra parte, se puede decidir si se quiere que se incluyan comentarios sobre el código generado, o bien obviarlo.

Paso 6. Nombres y tipos de las clases y de los ficheros creados.

En este último paso (ver Figura G), es posible elegir los nombres a asignar a las diferentes clases, ficheros de código fuente y ficheros de cabecera. Además, en las clases para generar la vista del documento, también se puede elegir la clase padre de la cual se derivará la propia. En este caso, debe seleccionarse la clase *CEditView*, al querer crear una aplicación para edición de textos ASCII.

Mediante AppWizard es posible elegir un tipo de aplicación entre diferentes opciones, de la forma más sencilla e intuitiva posible

UN VISTAZO AL CÓDIGO GENERADO

Se generan automáticamente varios ficheros de código fuente, cada uno con una labor bien diferenciada y basada en cada una de las clases principales de las aplicaciones creadas por *AppWizard* como son *CWinApp*, *CDocument*, *Cview* y *CFrameWnd*. Para el ejemplo, el fichero *MFCEditor.cpp* contiene el código encargado de crear el único objeto derivado de *CWinApp* que debe existir, inicializar la aplicación dentro de la función miembro *InitInstance*. Además contiene la clase para visualizar la ventana de "Acerca de...".

Otro fichero es *MainFrm.cpp*, donde se incluye la clase derivada de *CFrameWnd*, encargada de gestionar la creación y control de la ventana principal. Dentro de este fichero se crean además, la barra de herramientas y la barra de estado correspondiente, y es aquí donde se deben implementar los métodos para manejar la ventana principal. Los otros dos ficheros generados son *MCFEditorDoc.cpp* y *MFCEditorView* que incluyen las clases (y sus correspondientes métodos) para crear, manejar y asociar cada documento con la visualización del mismo dentro de la ventana principal. Todos los recursos creados se almacenan en el fichero *MFCEditor.rc*, donde se incluyen todos los recursos necesarios para el funcionamiento del esqueleto de la aplicación.

Aunque parezca un tanto increíble, este código es compilable sin ningún tipo de error y da como resultado a una aplicación totalmente operativa con las funciones típicas de un editor de texto (ver Figura G).

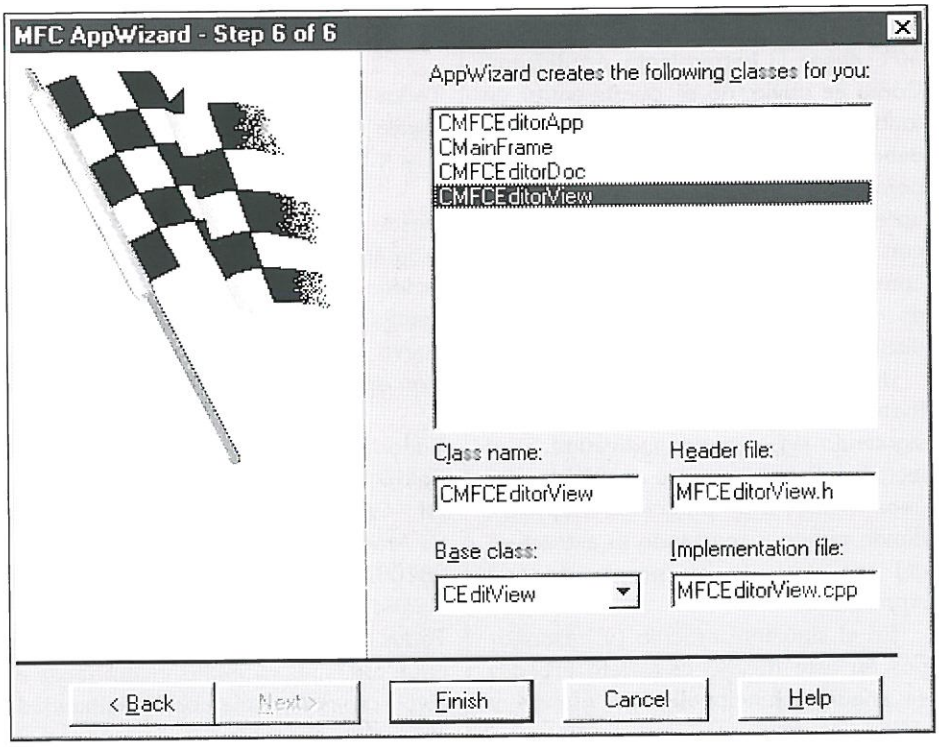


Figura F. Nombres y tipos de clases y ficheros generados.

LA CLASE CWINAPP

CWinApp es una clase que encapsula los procesos de inicialización, ejecución y terminación de una aplicación Windows. Sólo puede existir un objeto de esta clase en cada aplicación y debe ser creado con anterioridad a cualquier ventana.

En *Win32* y al ser un sistema con múltiples hilos de ejecución, la clase *CWinApp* es el *thread* primario de la aplicación, aunque no tiene porque ser el único.

En una aplicación basada en la *MFC* también existe un punto de entrada en la aplicación del tipo *WinMain*, aunque en este entorno queda encapsulado en la librería. La secuencia de ejecución típica es la siguiente. *WinMain* llama a *InitInstance*, función que debe ser sobrecargada por el programador para

iniciar las operaciones particulares de cada aplicación. Después de salir de *InitInstance*, se llama a la función miembro *Run*, la cual ejecuta el bucle de mensajes y permite en los tiempos muertos ejecutar acciones en segundo plano (*background*) sobrecargando la función miembro *OnIdle*. Cuando se sale del bucle de mensajes al acabar la aplicación, se llama a la función *ExitInstance*, por lo que de nuevo, y en caso de ser necesario, se puede sobrecargar dicha función para implementar las operaciones debidas.

LA CLASE CWND Y LAS VENTANAS WINDOWS

Uno de los puntos fundamentales para programar con la *MFC* es tener claro las diferencias entre una clase de ventana de C++ y una clase de ventana de Windows. La clase *CWnd*, y derivadas, es una clase C++ creada directamente desde el propio programa, y una ventana Windows es un *handle* a una estructura interna de Windows, que se corresponde con una ventana en pantalla y que consume recursos del sistema.

Cuando se crea una clase de ventana mediante el constructor, no se crea la ventana Windows hasta que no se realiza una llamada a la función miembro *Create* de dicha clase. Después de esta llamada y suponiendo un resultado positivo de dicha creación, se guarda en la variable *m_hWnd* el *handle* de acceso a la ventana Windows.

En la clase *CWnd* se encapsulan algunas de las llamadas al API que están de algún modo relacionadas con las ventanas Windows, como enviar mensajes, conocer el estado de la ventana, convertir coordenadas, hacer scroll, actualizar el contenido y algunas otras operaciones. Además, la clase *CWnd* ofrece una serie de funciones para atender a los diferentes mensajes recibidos por la ventana, con una estructura del siguiente modo. Ante unos mensajes del modo *WM_PAINT* o *WM_MOUSEMOVE*, la función que atiende a este mensaje es *OnPaint* o *OnMouseMove*, respectivamente.

Cualquier clase de ventana creada, cuando su ventana Windows correspondiente es destruida, debe ser eliminada con el fin de liberar la memoria reservada. Por ejemplo, si se tiene una

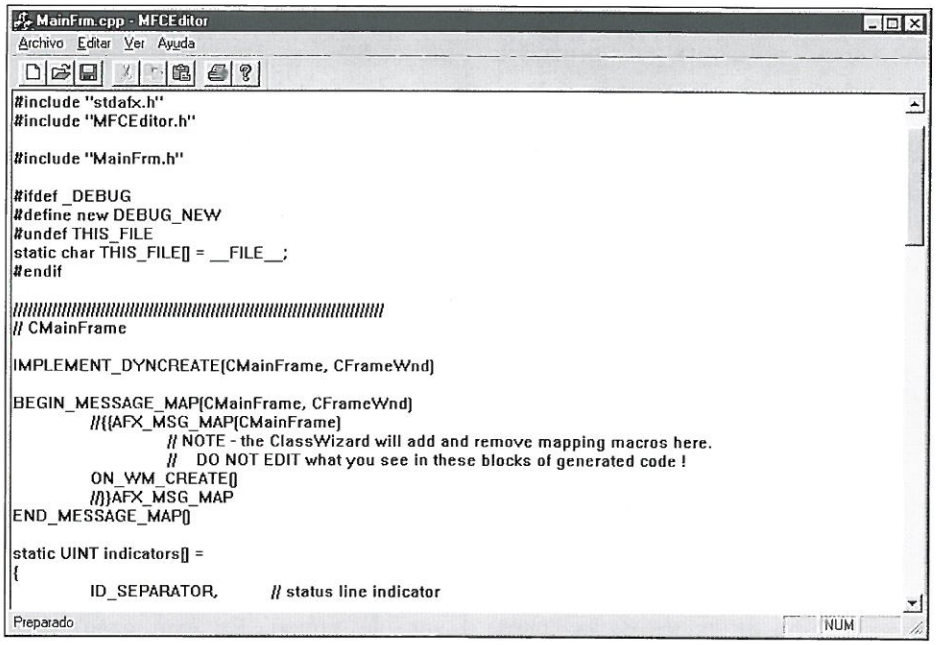


Figura G. Aspecto del editor generado con AppWizard.

ventana y le llega un mensaje del tipo *WM_CLOSE*, el manejador por defecto llama a la función *DestroyWindow*, dando como resultado a la destrucción de la ventana Windows. Posteriormente, se debe llamar al operador *delete* para liberar la memoria reservada con *new* para dicha clase.

CONCLUSIÓN

En este articulo se ha pretendido sentar los conceptos básicos para iniciar una aplicación basada en la librería *MFC* desde la herramienta *AppWizard*. Como es obvio, no se puede entrar en todos los detalles que se quisieran, y se deben pasar por alto algunos detalles, pero como siempre, la mejor forma de aprender es haciendo muchas pruebas con diferentes tipos de aplicaciones, comprobando los cambios y observando cual es la mejor solución para las distintas necesidades.

Junto al articulo se incluyen los fuentes creados por *AppWizard*, siguiendo las opciones indicadas. Si el lector realiza un proceso similar, tendrá unos fuentes muy parecidos o iguales, donde puede ir revisando la estructura del esqueleto de los programas con *AppWizard*.

En el siguiente articulo (y basándonos en este mismo esqueleto creado) se añadirán funcionalidades nuevas y diferentes elementos, para ilustrar el modo de moverse por las diferentes clases del programa mediante las herramientas *ClassWizard* y *ClassBar*.

Para contactar con el autor:

Desde Internet:
E-Mail: jjrio@ibm.net
jjrio@msn.com

Desde MSN: jjrio

CONCURSO SOLO PROGRAMADORES Y MICROSOFT VISUAL C++ 4.0

Todos los meses Sólo Programadores regala a sus lectores Microsoft Visual C++ 4.0 :

Desde la redacción de Sólo Programadores animamos a todos los lectores a enviarnos todo tipo de preguntas, dudas o comentarios sobre este nuevo curso dedicado a la programación en Visual C++ 4.0 a:

Editorial Tower
Sólo Programadores
ref " Concurso Visual C++ 4.0 "
C/ Aragoneses 7
28100 Pol. Ind. Alcobendas
Madrid

Todos los meses se elegirá una carta entre todas las recibidas valorándose el nivel de conocimientos, originalidad, etc. y se obsequiará a su autor con esta magnífica herramienta.

CONTENIDO DEL CD-ROM

RESULTADO CONCURSO DE DEMOS

Este mes, junto a la revista se incluye un CD-ROM de regalo. El principal contenido del mismo son gran parte de las Demos llegadas a la redacción para participar en el Primer Concurso de Demos e Infografía. Lamentablemente, otras se han quedado fuera del CD por diversos motivos, principalmente porque colgaran el ordenador.

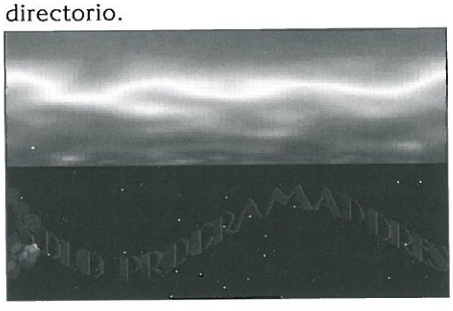
También se ha incluido en el CD una completa colección de herramientas relacionadas con la serie de artículos sobre WEB y el lenguaje HTML.

Otra de las partes interesantes la compone el espacio dedicado al Shareware enviado por los lectores a la redacción. Desde estas líneas animamos a que todos aquellos lectores que lo deseen continúen enviando sus programas que gustosamente serán incluidos en próximos CDs.

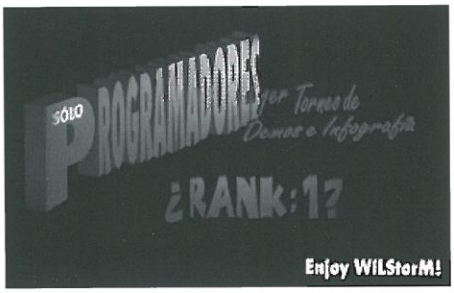
Por último, también es posible encontrar los ficheros que habitualmente adjuntamos con la revista con los programas ejemplo correspondientes a los distintos artículos.

CONCURSO

Este directorio contiene las demos del concurso agrupadas por categoras. Dentro de éste, las producciones de una determinada categoría están agrupadas en el correspondiente subdirectorio.



No se ha hecho ningún filtro en cuanto a la calidad de las mismas, pues la intencin al organizar este concurso fue animar a todos los lectores a programar. Desde estas lneas queremos expresar nuestro agradecimiento a todos aquellos que han participado, y nuestra felicitacin a los creadores de las mejores producciones. Ha sido una experiencia gratificante el comprobar la gran calidad y profesionalidad de algunos de los trabajos recibidos.



OBSERVACIONES PARA LA EJECUCIÓN DE LAS DEMOS:

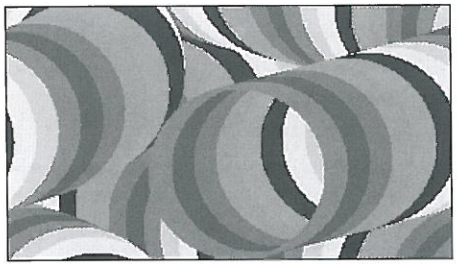
Demos

NOMBRE	EJECUTABLE	CD	EMM386
BOLAS	Bolas.exe	SÍ	SÍ
CULTOMSX	Cultomsx.exe	SÍ	SÍ
DJ	D&J.exe	SÍ	SÍ
DSHADOWS	DShadows.exe	No	SÍ
GEMINIS	Geminis.exe	SÍ	SÍ
ELEMENT	ELEMENT.COM	SÍ	SÍ
PARANOIA	Paranoia.exe	SÍ	SÍ
PUAJJJ	run.bat	No	SÍ
REAL01	Run.bat	SÍ	SÍ
SHADOW	Shadow.exe	SÍ	SÍ
SIMPLE	Simple.exe	No	SÍ
SUEÑO_C	caos.exe	SÍ	SÍ
THUNDER	Thunder.exe	SÍ	No
UNNA	Unna.exe	SÍ	SÍ
XMAS94	runme.bat	SÍ	SÍ

INTROS 64KB

NOMBRE	EJECUTABLE	CD	EMM386
1STRO	1STRO.EXE	SÍ	SÍ
ASTURNET	ASTURNET.EXE	SÍ	SÍ
LAI	LAI.EXE	SÍ	SÍ
REVERSE	REVERSE.EXE	SÍ	SÍ
SCRWIL	SCRWIL!.EXE	SÍ	SÍ
WPLASMA2	GO.BAT	SÍ	SÍ

Pero primero es conveniente dar un aviso importante. Algunas producciones, sobre todo en la categora de animaciones, no se pueden ejecutar desde el CD-ROM, por lo tanto no garantizamos el funcionamiento de las demos si no son copiadas previamente a disco duro. El mejor consejo, por tanto, es instalarse en el disco duro todas las demos para poder visionarlas sin problemas. Esto es debido a que, por ejemplo, en la categora de animaciones, el Aniplay.exe intenta crear un fichero de memoria virtual en el CD, con el consiguiente fallo.



NOMBRE	EJECUTABLE	CD
ANI	ANIPLAY	Sí
BASE	ANIPLAY	Sí
DRAGON	RUN.BAT	Sí
ESPEJO	ANIPLAY	No
FUTBOLIN	FUTBOLIN.BAT	No
FUTURE	ANIPLAY	No
GRAMOLA	GRAMOLA.BAT	Sí
HEY	(*)	Sí
LASERFX	ANIPLAY	Sí
MOUNTAIN	MOUNTAIN.BAT	Sí
ODISEA	ODISEA.BAT	Sí
PC	PC.BAT	Sí
PERSISTE	ANIMA.BAT	Sí
PIRAMIDE	(*)	Sí
POSTAL	(*)	Sí
PRESENTA	ANIPLAY	Sí
RAILAIR	ANIPLAY	No
ROBOT	(*)	Sí
SALTO	SALTO.BAT	Sí
SHE_POWE	ANIPLAY	No
SIACOM	SIACOM.BAT	Sí
SPHERE	FLI2VGA.EXE	Sí
VIR	FLI2VGA.EXE	Sí

(*) Estos ficheros son AVIs y deben ser visionados mediante un visualizador como Quickview o con el Transmisor de medios de Windows

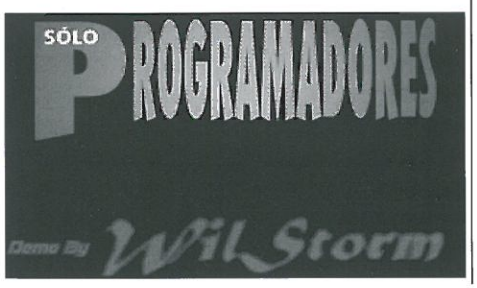
INTROS 4K

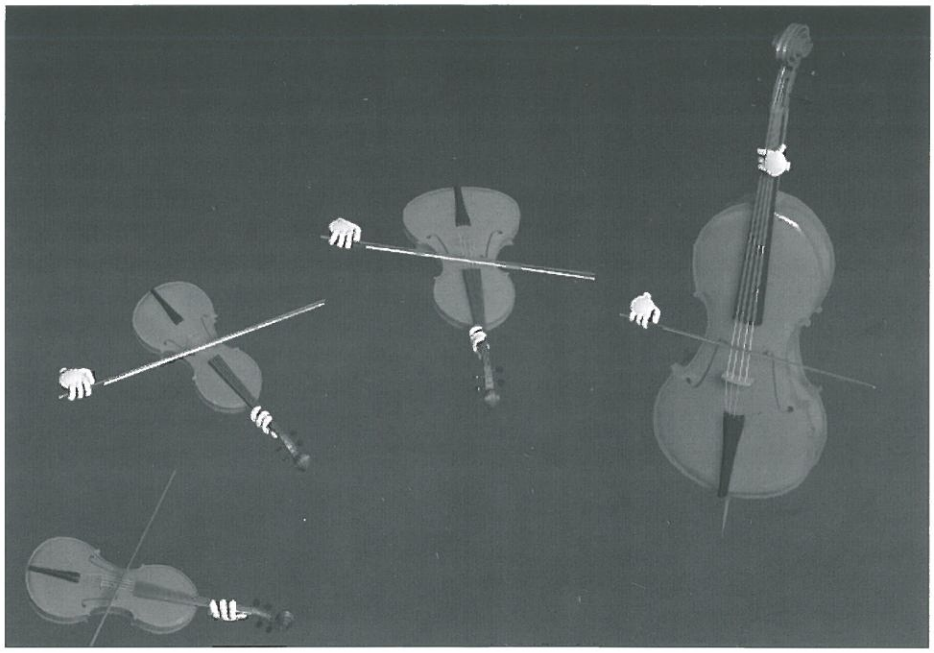
En este directorio se encuentran las intros de tamao menor o igual a 4k, todas pueden ser ejecutadas desde CD sin necesidad de liberar memoria excepto la denominada Tunel que necesita ser instalada a disco duro con un arranque totalmente limpio.

IMÁGENES

Colección de imágenes seleccionadas entre las concursantes, para su visualizacin utilizar alguna utilidad como GWS, VPic, Alchemy, etc.

Morphing: SINFORMA
Sin sonido funciona desde el CD, ejecutando NOSOUND.BAT. Para configurar el sonido es necesario copiar los ficheros al disco duro y ejecutar START.BAT





GANADORES

Seguro que muchos lectores lo primero que desearán comprobar es la calidad de las producciones ganadoras de cada categora. Tras un largo proceso de evaluación, el Jurado de Sólo Programadores consideró que los mejores en cada categoría son los que se indican a continuación:

Categoría	Sección	Autor	Título
Demos	Intros 4 Kb	Marathon Man	RC-Mania
	Intros 64 Kb	Ispania	Lal
	Demos	Ispania	Puajjj
Infografía	Imagen fija	XIMO C.'95	Combate
	Animación	E. San Pedro	Hey

SHARE

En este directorio se incluyen algunos de los programas y utilidades llegados a la redaccin. Aprovechando estas líneas reiteramos nuestro ánimo a todos los programadores para que continúen enviando este tipo de programas. Una sugerencia al respecto es que se incluya documentación sobre los mismos. Por último, indicar que si alguno de estos programas resulta realmente interesante desde el punto de vista de su programacin se publicará en la revista un artculo especial comentando la utilidad.

Dentro de la muestra que se incluye cabe destacar:

3DBIBLIO

Se trata de una biblioteca de funciones gráficas para C. Con ella es posible obtener mayores resoluciones con la tarjeta VGA (de 256k), así como incluir objetos tridimensionales en los programas.

DENVIDEO

Es una librera realizada en ensamblador para poder utilizar modos X desde programas realizados en C.

SEGURO

Seguro 3.3 es una librera para la proteccin de aplicaciones realizadas en Clipper 5.x. Su utilización posibilita el evitar que las aplicaciones puedan ser copiadas.

WWW

Lo prometido es deuda, y en este directorio se encuentran diversas utilidades para los lectores de la serie de artículos sobre WEB y lenguaje HTML, desde Browsers, editores de páginas WEB hasta diversidad de informacin técnica y FAQs (Frequently Asked Questions) sobre WWW y HTML.

En el subdirectorio BROWSERS \WINDOWS se encuentran entre otros Netscape 1.0, Winweb browser, Cello browser, además de Netscape para Windows NT y Windows 95 y Linux.

- En el subdirectorio SERVERS se encuentran servidores de Web con fuentes, e incluso utilidades para recuperar datos de un form.

DISK20

Como habitualmente se realiza todo los meses, en este directorio se encuentran los fuentes correspondientes a los artculos publicados en este número 20 de Sólo Programadores.

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

Hola, mi consulta es referente al entorno de desarrollo GNU C++ para MS-DOS que se incluía en el CD-ROM del número 16 de Sólo Programadores. Quisiera conocer algo más sobre este producto: donde es desarrollado y por quien, y si existen versiones más recientes del mismo. ¿Está disponible para Windows 95?, ¿como puedo conseguir una copia? Un saludo a toda la redacción.

Rubén D'Toste
(Barcelona)

Gracias por el saludo. El nombre de software GNU nació con el Linux y casi todo lo desarrollado bajo esta denominación es para dicho Sistema Operativo. Sin embargo, existe multitud de aplicaciones para otros sistemas que también están creados con la misma filosofía. Una de ellas es la que nos ocupa : *DJGPP GNU C++*. Las personas que lo crearon (gente alrededor de todo el mundo, siempre sin ánimo de lucro) están coordinados por una especie de asociación denominada *Free Software Foundation*. Su ética se resume con las siguientes palabras : "Nosotros distribuimos el software con la esperanza de que sea útil, pero sin ningún tipo de garantía". La última versión es la 2.0, y como parece que el MS-DOS está un poco de capa caída (esperemos que no para siempre), pues a lo mejor tardan mucho es sacar otras versiones.

Sí que existe una versión para desarrollar bajo Windows 95 (o para Windows NT), se denomina *GNU-WIN32*. Aunque está todavía en fase de desarrollo, las herramientas están completamente creadas (*gcc*, *gdb*, *gas*, *et al*) y el proyecto se encuentra actualmente en la parte de creación y codificación de las librerías necesarias para poder acompañar a los compiladores. El objetivo es poder recompilar cualquier fuente GNU para Unix o Linux, y obtener código de 32 bits para Windows.
Se puede obtener cualquier versión de software GNU en multitud de sitios en Internet (su forma oficial de distribución).
Para cualquier software relacionado con GNU, probar con:

http://www.delorie.com/gnu/

y para el software pensado para Windows conectar con :

ftp.cygnus.com:/pub/gnu-win32

De todas formas, se está pensando en volver a regalar con otro número de la revista, la última versión y para los últimos Sistemas Operativos.

A ver si pueden responderme a las siguientes preguntas: De los entornos que han regalado con la revista, ¿qué compilador recomendáis para iniciarse a programar en C?: Symantec Profesional 6.0 o GNU C++. ¿Cual de ellos da más prestaciones?

Tomás Muñoz Martín
(Barcelona)

Si lo que busca es empezar a conocer el mundo de la programación en C, no debe fijarse en las prestaciones. Elija el que tenga un entorno integrado más agradable, con las opciones básicas de generación más accesibles y documentadas, pero sin renunciar a los parámetros de compilación más potentes. Es importante que genere ejecutables para distintas plataformas y sistemas operativos, pues de esta forma, se puede practicar y probar de muy distintas maneras.
Además, las prestaciones hoy en día, son muy similares entre los distintos compiladores; y no se comercializa ninguno que no reúna unas características excepcionales.

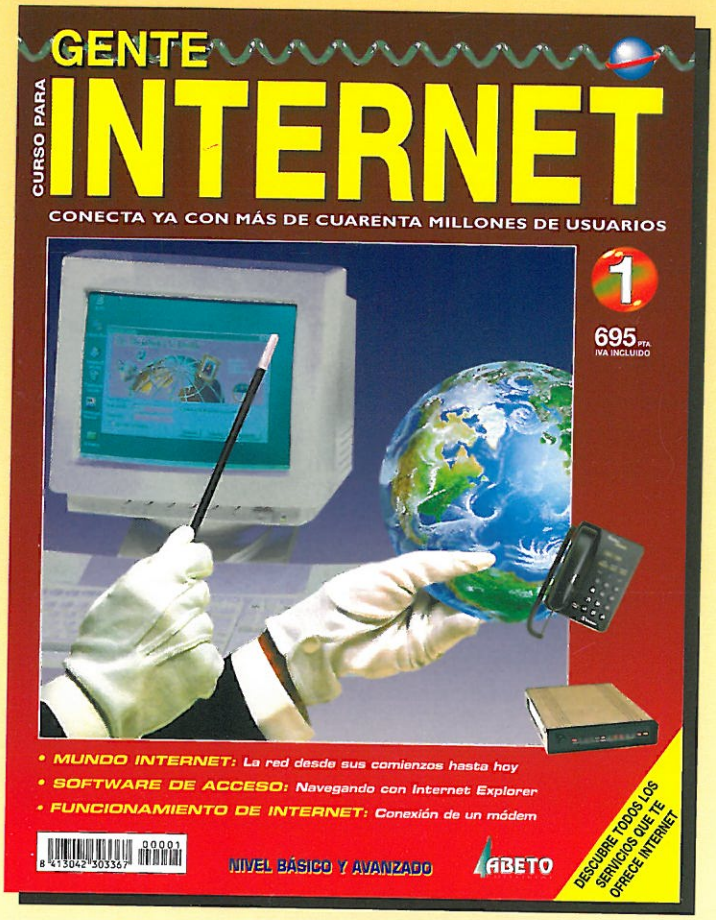
En los últimos tiempos se está hablando constantemente de Sistemas Operativos de 32 bits: que duplican las prestaciones, que si el núcleo es sólo de 16 bits, pero ejecuta código de 32 bits, que funciona en Modo Protegido... ¿Porqué son tan importantes (de cara a las prestaciones) los famosos 32 bits? Gracias y a seguir con la revista.

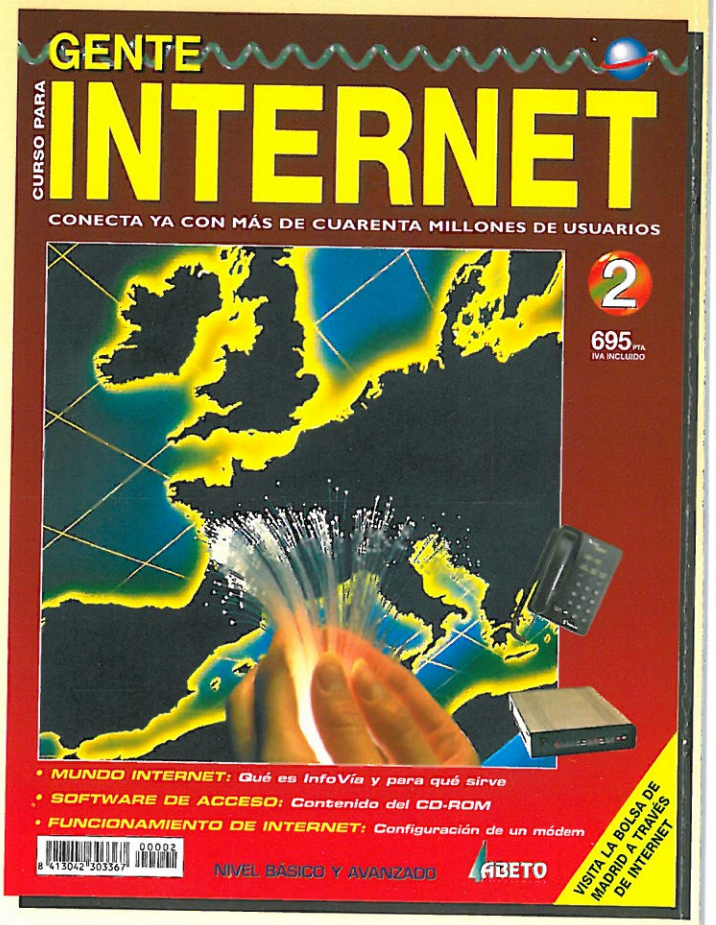
Enrique Martínez Guillén
(Madrid)

Existen varias y muy importantes razones para que un Sistema Operativo de 32 bits,

¡¡DOMINA YA INTERNET CON EL COLECCIONABLE GENTE INTERNET!!

15 DÍAS DE CONEXIÓN GRATIS


GENTE
CURSO PARA
INTERNET
CONECTA YA CON MÁS DE CUARENTA MILLONES DE USUARIOS
1
695 PTA. IVA INCLUIDO
• MUNDO INTERNET: La red desde sus comienzos hasta hoy
• SOFTWARE DE ACCESO: Navegando con Internet Explorer
• FUNCIONAMIENTO DE INTERNET: Conexión de un módem
8 413042 303367 00001
NIVEL BÁSICO Y AVANZADO
ABETO
DESCUBRE TODOS LOS SERVICIOS QUE TE OFRECE INTERNET


GENTE
CURSO PARA
INTERNET
CONECTA YA CON MÁS DE CUARENTA MILLONES DE USUARIOS
2
695 PTA. IVA INCLUIDO
• MUNDO INTERNET: Qué es InfoVía y para qué sirve
• SOFTWARE DE ACCESO: Contenido del CD-ROM
• FUNCIONAMIENTO DE INTERNET: Configuración de un módem
8 413042 303367 00002
NIVEL BÁSICO Y AVANZADO
ABETO
VISITA LA BOLSA DE MADRID A TRAVÉS DE INTERNET

INTERNET ES PARA TODOS

- ABOGADOS • PROFESIONALES
- EMPRESARIOS • PERIODISTAS
- MÉDICOS • INGENIEROS • ARQUITECTOS
- ESTUDIANTES • ECONOMISTAS
- USUARIOS EN GENERAL...

TODAS LAS SEMANAS EN TU QUIOSCO

LA COLECCIÓN CONSTA DE 40 FASCÍCULOS, 39 DISQUETES, 1 CD-ROM Y 4 TAPAS

ABETO EDITORIAL c/Aragoneses, 7 - Pol. Ind. Alcobendas - 28100 MADRID Tel.: (91) 661 42 11 - Fax (91) 661 43 86

SERVICIO 24 HORAS 906 - 31 32 3

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

Hola, mi consulta es referente al entorno de desarrollo GNU C++ para MS-DOS que se incluía en el CD-ROM del número 16 de Sólo Programadores. Quisiera conocer algo más sobre este producto: donde es desarrollado y por quien, y si existen versiones más recientes del mismo. ¿Está disponible para Windows 95?, ¿como puedo conseguir una copia? Un saludo a toda la redacción.

Rubén D'Toste
(Barcelona)

Gracias por el saludo. El nombre de software GNU nació con el Linux y casi todo lo desarrollado bajo esta denominación es para dicho Sistema Operativo. Sin embargo, existe multitud de aplicaciones para otros sistemas que también están creados con la misma filosofía. Una de ellas es la que nos ocupa : *DJGPP GNU C++*. Las personas que lo crearon (gente alrededor de todo el mundo, siempre sin ánimo de lucro) están coordinados por una especie de asociación denominada *Free Software Foundation*. Su ética se resume con las siguientes palabras : "Nosotros distribuimos el software con la esperanza de que sea útil, pero sin ningún tipo de garantía". La última versión es la 2.0, y como parece que el MS-DOS está un poco de capa caída (esperemos que no para siempre), pues a lo mejor tardan mucho es sacar otras versiones.

Sí que existe una versión para desarrollar bajo Windows 95 (o para Windows NT), se denomina *GNU-WIN32*. Aunque está todavía en fase de desarrollo, las herramientas están completamente creadas (*gcc, gdb, gas, et al*) y el proyecto se encuentra actualmente en la parte de creación y codificación de las librerías necesarias para poder acompañar a los compiladores. El objetivo es poder recompilar cualquier fuente GNU para Unix o Linux, y obtener código de 32 bits para Windows.
Se puede obtener cualquier versión de software GNU en multitud de sitios en Internet (su forma oficial de distribución).
Para cualquier software relacionado con GNU, probar con:

http://www.delorie.com/gnu/

y para el software pensado para Windows conectar con :

ftp.cygnus.com:/pub/gnu-win32

De todas formas, se está pensando en volver a regalar con otro número de la revista, la última versión y para los últimos Sistemas Operativos.

A ver si pueden responderme a las siguientes preguntas: De los entornos que han regalado con la revista, ¿qué compilador recomendáis para iniciarse a programar en C?: Symantec Profesional 6.0 o GNU C++. ¿Cual de ellos da más prestaciones?

Tomás Muñoz Martín
(Barcelona)

Si lo que busca es empezar a conocer el mundo de la programación en C, no debe fijarse en las prestaciones. Elija el que tenga un entorno integrado más agradable, con las opciones básicas de generación más accesibles y documentadas, pero sin renunciar a los parámetros de compilación más potentes. Es importante que genere ejecutables para distintas plataformas y sistemas operativos, pues de esta forma, se puede practicar y probar de muy distintas maneras.
Además, las prestaciones hoy en día, son muy similares entre los distintos compiladores; y no se comercializa ninguno que no reúna unas características excepcionales.

En los últimos tiempos se está hablando constantemente de Sistemas Operativos de 32 bits: que duplican las prestaciones, que si el núcleo es sólo de 16 bits, pero ejecuta código de 32 bits, que funciona en Modo Protegido... ¿Porqué son tan importantes (de cara a las prestaciones) los famosos 32 bits? Gracias y a seguir con la revista.

Enrique Martínez Guillén
(Madrid)

Existen varias y muy importantes razones para que un Sistema Operativo de 32 bits,

duplique (aunque no se llega en la práctica) a uno "convencional" de 16 bits :

Se libera de la lenta y obsoleta paginación : Al estar el microprocesador en Modo Protegido, se elimina el límite de los 64 Kbytes direccionables directamente por el procesador, ahorrándose la constante comprobación de si se ha sobrepasado el límite o no, y pudiéndose copiar o mover enormes bloques de memoria directamente.

Se dispone de más registros internos del microprocesador (y el doble de grandes) : Se mejoran todas las operaciones que realiza el micro, y se puede mover el doble de datos (los 4 bytes de los 32 bits, contra los 2 bytes de los 16 bits) con el mismo consumo de tiempo de la CPU.

Se elimina la barrera de los 640 Kbytes de la memoria "convencional" : No es necesario recurrir a "trucos" para poder acceder a la memoria superior (memoria expandida, memoria extendida, *VCPI*, *drivers* de memoria, etc.).

Se mejora la multitarea y el control de procesos : Con el microprocesador activo en este modo, están disponibles todas las funciones y características para gestionar la multitarea, propias de los modernas máquinas.

Trabajo como programador en una empresa en la que tenemos instalada una gestión realizada en lenguaje Cobol. Tenemos interés en actualizar la gestión a un lenguaje que permita trabajar en un entorno gráfico (Windows u OS/2) y necesitamos que el cambio se realice de manera rápida y aprovechando los ficheros de datos que ya disponemos. Habíamos pensado en los emergentes lenguajes visuales, como Visual C++ o Delphi; pero no sabemos cual es la mejor opción.
En referencia a la conversión de los datos, ¿cómo se podrían convertir los ficheros indexados de Cobol a estos sistemas?

Ignacio Casasnovas Anglada
(Ciutadella de Menorca)

La mejor opción, depende de los conocimientos previos de los que se disponga. Si se conoce el lenguaje C : Visual C++ ; y si por el contrario se domina el Pascal : Delphi. En el caso de desenvolverse bien en cualquiera, es una cuestión de gustos, pues ambos son de una calidad excepcional.

Estos entornos, importan perfectamente ficheros ASCII o ficheros con el formato XBASE (Dbase, Foxpro, Clipper, etc). También es posible cargar los datos mediante la posibilidad de "Cortar y Pegar" de Windows, o utilizando la potencia del intercambio OLE. Para obtener datos utilizables desde los ficheros VSAM (o indexados) de Cobol, lo mejor es utilizar un sencillo conversor de estos ficheros a formato plano ASCII ; o crearse un sencillo programa "Cíbolo" que genere dichos ficheros ASCII en función de los datos originales.

A menudo nuestros lectores nos preguntan cómo se pueden conseguir los números atrasados, o cómo solicitar las fichas de programación publicadas conjuntamente e incluso cómo poder convertirse en colaborador de Sólo Programadores.

Pues bien, aquí se detalla la forma de realizarlo:

Por correo a la
siguiente dirección:

TOWER COMMUNICATIONS SRL
c/ Aragoneses, 7
28100 Pol. Ind. Alcobendas

Por fax al siguiente número:
(91) 66 14 386
Llamando al siguiente teléfono:
(91) 66 14 211
Horario de 9 a 14'00 y de 15:00 a 18:00 horas

Quisiera ejecutar un programa residente en un ordenador remoto desde un programa propio (algo similar a la orden spawn de C). ¿Es esto posible?, ¿existen librerías para ello? Muchas gracias de antemano.

Jesús María Fuentes Navarro (Toledo)

Realizar esa operación desarrollando un software propio es un trabajo enorme y, hoy en día, un poco descabellado. Lo mejor sería el utilizar cualquier paquete de comunicaciones existente que permita ejecutar programas y tareas llamando remotamente desde otro ordenador. Uno de los más potentes (y sencillo) es el Terminate, que regalamos en su versión freeware en esta revista hace unos meses. Terminate dispone de un modo host en el que permanece esperando a llamadas exteriores (en realidad se trata de una sencilla BBS) y según el usuario entrante, pues da ciertos derechos y posibilidades, como la de ejecutar programas o procesos batch mediante un menú de opciones. Con esto se ahorra todo el proceso del manejo del modem, del control de la conexión y del mantenimiento de errores en la transferencia.

Si los ordenadores a conectar estuvieran ya unidos por un cable serie, entonces no sería necesario ni Terminate, pues el mismo MS-DOS tiene peque-as utilidades para un control limitado de ordenadores remotos.

ENTRE EN EL MUNDO DE

SUPEROFERTA PARA SUSCRIPTORES Y LECTORES DE

SÓLO PROGRAMADORES
Revista práctica para usuarios de PC

+ 50% DESCUENTO en VISUAL C++

CON LAS HERRAMIENTAS VISUALES DE MICROSOFT

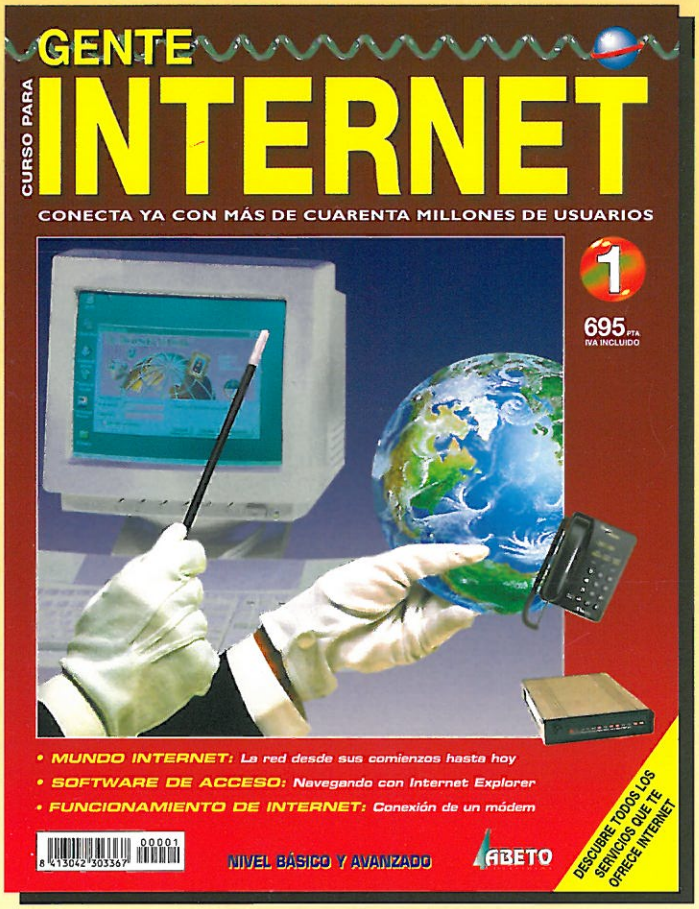
PARA BENEFICIARSE DE ESTA OFERTA LOS LECTORES NO SUSCRIPTORES DEBERÁN REMITIR LOS CUPONES QUE APARECERÁN EN ESTA PÁGINA

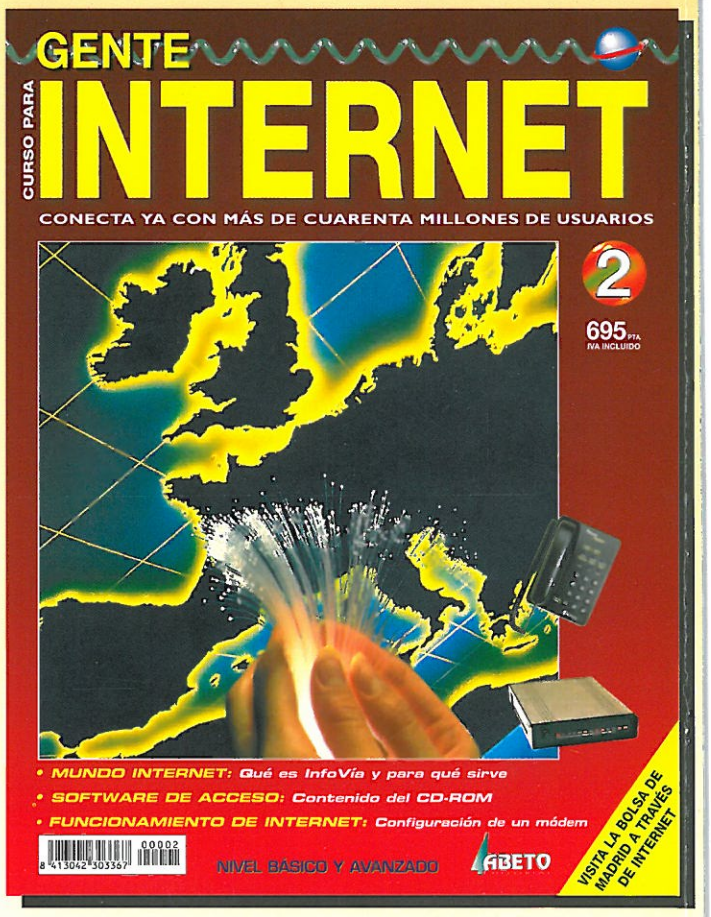
CURSO DE VISUAL C++

CUPÓN **3**

¡¡DOMINA YA INTERNET CON EL COLECCIONABLE GENTE INTERNET!!

15 DÍAS DE CONEXIÓN GRATIS


GENTE
CURSO PARA
INTERNET
CONECTA YA CON MÁS DE CUARENTA MILLONES DE USUARIOS
1
695 PTA. IVA INCLUIDO
• MUNDO INTERNET: La red desde sus comienzos hasta hoy
• SOFTWARE DE ACCESO: Navegando con Internet Explorer
• FUNCIONAMIENTO DE INTERNET: Conexión de un módem
NIVEL BÁSICO Y AVANZADO
ABETO
DESCUBRE TODOS LOS SERVICIOS QUE TE OFRECE INTERNET


GENTE
CURSO PARA
INTERNET
CONECTA YA CON MÁS DE CUARENTA MILLONES DE USUARIOS
2
695 PTA. IVA INCLUIDO
• MUNDO INTERNET: Qué es InfoVía y para qué sirve
• SOFTWARE DE ACCESO: Contenido del CD-ROM
• FUNCIONAMIENTO DE INTERNET: Configuración de un módem
NIVEL BÁSICO Y AVANZADO
ABETO
VISITA LA BOLSA DE MADRID A TRAVÉS DE INTERNET

INTERNET ES PARA TODOS

- ABOGADOS • PROFESIONALES
- EMPRESARIOS • PERIODISTAS
- MÉDICOS • INGENIEROS • ARQUITECTOS
- ESTUDIANTES • ECONOMISTAS
- USUARIOS EN GENERAL...

TODAS LAS SEMANAS EN TU QUIOSCO

LA COLECCIÓN CONSTA DE 40 FASCÍCULOS, 39 DISQUETES, 1 CD-ROM Y 4 TAPAS

ABETO EDITORIAL c/Aragoneses, 7 - Pol. Ind. Alcobendas - 28100 MADRID Tel.: (91) 661 42 11 - Fax (91) 661 43 86

SERVICIO 24 HORAS 906 - 31 32 3